Online Makespan Minimization with Parallel Schedules

Susanne Albers and Matthias Hellwig

Department of Computer Science, Humboldt-Universität zu Berlin
{albers,mhellwig}@informatik.hu-berlin.de

Abstract. Online makespan minimization is a classical problem in which a sequence of jobs $\sigma = J_1, \ldots, J_n$ has to be scheduled on m identical parallel machines so as to minimize the maximum completion time of any job. In this paper we investigate the problem with an essentially new model of resource augmentation. More specifically, an online algorithm is allowed to build several schedules in parallel while processing σ. At the end of the scheduling process the best schedule is selected. This model can be viewed as providing an online algorithm with extra space, which is invested to maintain multiple solutions. The setting is of particular interest in parallel processing environments where each processor can maintain a single or a small set of solutions.

As a main result we develop a $(4/3 + \varepsilon)$-competitive algorithm, for any $0 < \varepsilon \leq 1$, that uses a constant number of schedules. The constant is $1/\varepsilon^{O(\log(1/\varepsilon))}$. We also give a $(1 + \varepsilon)$-competitive algorithm, for any $0 < \varepsilon \leq 1$, that builds a polynomial number of $(m/\varepsilon)^{O(\log(1/\varepsilon)/\varepsilon)}$ schedules. This value depends on m but is independent of the input σ. The performance guarantees are nearly best possible. We show that any algorithm that achieves a competitiveness smaller than $4/3$ must construct $\Omega(m)$ schedules. Our algorithms make use of novel guessing schemes that (1) predict the optimum makespan of a job sequence σ to within a factor of $1+\varepsilon$ and (2) guess the job processing times and their frequencies in σ. In (2) we have to sparsify the universe of all guesses so as to reduce the number of schedules to a constant.

The competitive ratios achieved using parallel schedules are considerably smaller than those in the standard problem without resource augmentation. Furthermore they are at least as good and in most cases better than the ratios obtained with other means of resource augmentation for makespan minimization.

1 Introduction

Makespan minimization is a fundamental and extensively studied problem in scheduling theory. Consider a sequence of jobs $\sigma = J_1, \ldots, J_n$ that has to be scheduled on m identical parallel machines. Each job J_t is specified by a processing time $p_t > 0$, $1 \leq t \leq n$. Preemption of jobs is not allowed. The goal is to minimize the makespan, i. e. the maximum completion time of any job in the constructed schedule. We focus on the online version of the problem where the jobs of σ arrive one by one. Each incoming job J_t has to be assigned immediately to one of the machines without knowledge of any future jobs $J_{t'}$, $t' > t$.

Online algorithms for makespan minimization have been studied since the 1960s. In an early paper Graham [21] showed that the famous *List* scheduling algorithm is $(2 - 1/m)$-competitive. The best online strategy currently known achieves a competitiveness of about 1.92. Makespan minimization has also been studied with various types

of *resource augmentation*, giving an online algorithm additional information or power while processing σ. The following scenarios were considered. (1) An online algorithm knows the optimum makespan or the sum of the processing times of σ. (2) An online strategy has a buffer that can be used to reorder σ. Whenever a job arrives, it is inserted into the buffer; then one job of the buffer is removed and placed in the current schedule. (3) An online algorithm may migrate a certain number or volume of jobs.

In this paper we investigate makespan minimization assuming that an online algorithm is allowed to build several schedules in parallel while processing a job sequence σ. Each incoming job is sequenced in each of the schedules. At the end of the scheduling process the best schedule is selected. We believe that this is a natural form of resource augmentation: In classical online makespan minimization, studied in the literature so far, an algorithm constructs a schedule while jobs arrive one by one. Once all jobs have arrived, the schedule may be executed. Hence in this standard framework there is a priori no reason why an algorithm should not be able to construct several solutions, the best of which is finally chosen.

Our new proposed setting can be viewed as providing an online algorithm with extra space, which is used to maintain several solutions. Very little is known about the value of extra space in the design of online algorithms. Makespan minimization with parallel schedules is of particular interest in parallel processing environments where each processor can take care of a single or a small set of schedules. We develop algorithms that require hardly any coordination or communication among the schedules. Last not least the proposed setting is interesting w. r. t. to the foundations of scheduling theory, giving insight into the value of multiple candidate solutions.

Makespan minimization with parallel schedules was also addressed by Kellerer et al. [27]. However, the paper focused on the restricted setting with $m = 2$ machines. In this paper we explore the problem for a general number m of machines. As a main result we show that a constant number of schedules suffices to achieve a significantly improved competitiveness, compared to the standard setting without resource augmentation. The competitive ratios obtained are at least as good and in most cases better than those attained in the other models of resource augmentation mentioned above.

The approach to grant an online algorithm extra space, invested to maintain multiple solutions, could be interesting in other problems as well. The approach is viable in applications where an online algorithm constructs a solution that is used when the entire input has arrived. This is the case, for instance, in basic online graph coloring and matching problems [24,26,29]. The approach is also promising in problems that can be solved by a set of independent agents, each of which constructs a separate solution. Good examples are online navigation and exploration problems in robotics [11,12,14]. Some results are known for graph search and exploration, see e. g. [10,19,28], but the approach has not been studied for geometric environments.

Problem definition: We investigate the problem *Makespan Minimization with Parallel Schedules (MPS)*. As always, the jobs of a sequence $\sigma = J_1, \ldots, J_n$ arrive one by one and must be scheduled non-preemptively on m identical parallel machines. Each job J_t has a processing time $p_t > 0$. In MPS, an online algorithm $\mathcal{A}$ may maintain a set $\mathcal{S} = \{S_1, \ldots, S_l\}$ of schedules during the scheduling process while jobs of σ arrive. Each job J_t is sequenced in each schedule S_k, $1 \leq k \leq l$. At the end of σ, algorithm $\mathcal{A}$

selects a schedule $S_k \in \mathcal{S}$ having the smallest makespan and outputs this solution. The other schedules of $\mathcal{S}$ are deleted.

As we shall show MPS can be reduced to the problem variant where the optimum makespan of the job sequence to the processed is known in advance. Hence let MPS_{opt} denote the variant of MPS where, prior to the arrival of the first job, an algorithm $\mathcal{A}$ is given the value of the optimum makespan $\text{OPT}(\sigma)$ for the incoming job sequence σ. An algorithm $\mathcal{A}$ for MPS or MPS_{opt} is ρ-competitive if, for every job sequence σ, it outputs a schedule whose makespan is at most ρ times $\text{OPT}(\sigma)$.

Our contribution: We present a comprehensive study of MPS. We develop a $(4/3+\varepsilon)$-competitive algorithm, for any $0 < \varepsilon \le 1$, using a constant number of $1/\varepsilon^{O(\log(1/\varepsilon))}$ schedules. Furthermore, we give a $(1+\varepsilon)$-competitive algorithm, for any $0 < \varepsilon \le 1$, that uses a polynomial number of schedules. The number is $(m/\varepsilon)^{O(\log(1/\varepsilon)/\varepsilon)}$, which depends on m but is independent of the job sequence σ. These performance guarantees are nearly best possible. The algorithms are obtained via some intermediate results that may be of independent interest.

First, in Section 2 we show that the original problem MPS can be reduced to the variant MPS_{opt} in which the optimum makespan is known. More specifically, given any ρ-competitive algorithm $\mathcal{A}$ for MPS_{opt} we construct a $(\rho+\varepsilon)$-competitive algorithm $\mathcal{A}^*(\varepsilon)$, for any $0 < \varepsilon \le 1$. If $\mathcal{A}$ uses l schedules, then $\mathcal{A}^*(\varepsilon)$ uses $l \cdot \lceil \log(1+\frac{6\rho}{\varepsilon})/\log(1+\frac{\varepsilon}{3\rho})\rceil$ schedules. The construction works for any algorithm $\mathcal{A}$ for MPS_{opt}. In particular we could use a 1.6-competitive algorithm by Chen et al. [13] that assumes that the optimum makespan is known and builds a single schedule. We would obtain a $(1.6+\varepsilon)$-competitive algorithm that builds at most $\lceil \log(1+10/\varepsilon)/\log(1+\varepsilon/5)\rceil$ schedules.

We proceed and develop algorithms for MPS_{opt}. In Section 3 we give a $(1+\varepsilon)$-competitive algorithm, for any $0 < \varepsilon \le 1$, that uses $(\lfloor 2m/\varepsilon \rfloor + 1)^{\lceil \log(2/\varepsilon)/\log(1+\varepsilon/2)\rceil}$ schedules. In Section 4 we devise a $(4/3+\varepsilon)$-competitive algorithm, for any $0 < \varepsilon \le 1$, that uses $1/\varepsilon^{O(\log(1/\varepsilon))}$ schedules. Combining these algorithms with $\mathcal{A}^*(\varepsilon)$, we derive the two algorithms for MPS mentioned in the above paragraph; see also Section 5. The number of schedules used by our strategies depends on $1/\varepsilon$ and exponentially on $\log(1/\varepsilon)$ or $1/\varepsilon$. Such a dependence seems inherent if we wish to explore the full power of parallel schedules. The trade-offs resemble those exhibited by PTASes in offline approximation. Recall that the PTAS by Hochbaum and Shmoys [23] for makespan minimization achieves a $(1+\varepsilon)$-approximation with a running time of $O((n/\varepsilon)^{1/\varepsilon^2})$.

In Section 6 we present lower bounds. We show that any online algorithm for MPS that achieves a competitive ratio smaller than 4/3 must construct more than $\lfloor m/3 \rfloor$ schedules. Hence the competitive ratio of 4/3 is best possible using a constant number of schedules. We show a second lower bound that implies that the number of schedules of our $(1+\varepsilon)$-competitive algorithm is nearly optimal, up to a polynomial factor.

Our algorithms make use of novel guessing schemes. $\mathcal{A}^*(\varepsilon)$ works with guesses on the optimum makespan. Guessing and *doubling* the value of the optimal solution is a technique that has been applied in other load balancing problems, see e. g. [6]. However here we design a refined scheme that carefully sets and readjusts guesses so that the resulting competitive ratio increases by a factor of $1+\varepsilon$ only, for any $\varepsilon > 0$. Moreover, the readjustment and job assignment rules have to ensure that scheduling errors, made when guesses were to small, are not critical. Our $(4/3+\varepsilon)$-competitive algorithm works

with guesses on the job processing times and their frequencies in σ. In order to achieve a constant number of schedules, we have to sparsify the set of all possible guesses. As far as we know such an approach has not been used in the literature before.

All our algorithms have the property that the parallel schedules are constructed basically independently. The algorithms for $\mathrm{MPS}_{\mathrm{opt}}$ require no coordination at all among the schedules. In $\mathcal{A}^*(\varepsilon)$ a schedule only has to report when it fails, i. e. when a guess on the optimum makespan is too small.

The competitive ratios achieved with parallel schedules are considerably smaller than the best ratios of about 1.92 known for the scenario without resource augmentation. Our ratio of $(4/3+\varepsilon)$, for small ε, is lower than the competitiveness of about 1.46 obtained in the settings where a reordering buffer of size $O(m)$ is available or $O(m)$ jobs may be reassigned. Skutella et al. [33] gave an online algorithm that is $(1+\varepsilon)$-competitive if, before the assignment of any job J_t, jobs of processing volume $2^{O((1/\varepsilon)\log^2(1/\varepsilon))}p_t$ may be migrated. Hence the total amount of extra resources used while scheduling σ depends on the input sequence.

Related work: Makespan minimization with parallel schedules was first studied by Kellerer et al. [27]. They assume that $m=2$ machines are available and two schedules may be constructed. They show that in this case the optimal competitive ratio is 4/3.

We summarize results known for online makespan minimization without resource augmentation. As mentioned before, *List* is $(2-1/m)$-competitive. Deterministic online algorithms with a smaller competitive ratio were presented in [1,9,18,20,25]. The best algorithm currently known is 1.9201-competitive [18]. Lower bounds on the performance of deterministic strategies were given in [1,8,16,22,31,32]. The best bound currently known is 1.88, see [31]. No randomized online algorithm whose competitive ratio is provably below the deterministic lower bound is currently known for general m.

We next review the results for the various models of resource augmentation. Articles [3,4,5,7,13,27] study makespan minimization assuming that an online algorithm knows the optimum makespan or the sum of the processing times of σ. Chen et al. [13] developed a 1.6-competitive algorithm. Azar and Regev [7] showed that no online algorithm can attain a competitive ratio smaller than 4/3. The setting in which an online algorithm is given a reordering buffer was explored in [15,27]. Englert et al. [15] presented an algorithm that, using a buffer of size $O(m)$, achieves a competitive ratio of $W_{-1}(-1/e^2)/(1+W_{-1}(-1/e^2))\approx 1.46$, where W_{-1} is the Lambert W function. No algorithm using a buffer of size $o(n)$ can beat this ratio.

Makespan minimization with job migration was addressed in [2,33]. An algorithm that achieves again a competitiveness of $W_{-1}(-1/e^2)/(1+W_{-1}(-1/e^2))\approx 1.46$ and uses $O(m)$ job reassignments was devised in [2]. No algorithm using $o(n)$ reassignments can obtain a smaller competitiveness. Sanders et al. [33] study a scenario in which before the assignment of each job J_t, jobs up to a total processing volume of βp_i may be migrated, for some constant β. For $\beta=4/3$, they present a 1.5-competitive algorithm. They also show a $(1+\varepsilon)$-competitive algorithm, for any $\varepsilon>0$, where $\beta=2^{O((1/\varepsilon)\log^2(1/\varepsilon))}$.

As for memory in online algorithms, Sleator and Tarjan [34] studied the paging problem assuming that an online algorithm has a larger fast memory than an offline strategy. Raghavan and Snir [30] traded memory for randomness in online caching.

Notation: Throughout this paper it will be convenient to associate schedules with algorithms, i. e. a schedule S_k is maintained by an algorithm A_k that specifies how to assign jobs to machines in S_k. Thus an algorithm $\mathcal{A}$ for MPS or MPS$_{\text{opt}}$ can be viewed as a family $\{A_k\}_{k\in\mathcal{K}}$ of algorithms that maintain the various schedules. We will write $\mathcal{A} = \{A_k\}_{k\in\mathcal{K}}$. If $\mathcal{A}$ is an algorithm for MPS$_{\text{opt}}$, then the value $\text{OPT}(\sigma)$ is of course given to all algorithms of $\{A_k\}_{k\in\mathcal{K}}$. Furthermore, the *load* of a machine always denotes the sum of the processing times of the jobs already assigned to that machine.

2 Reducing MPS to MPS$_{\text{opt}}$

In this section we will show that any ρ-competitive algorithm $\mathcal{A}$ for MPS$_{\text{opt}}$ can be used to construct a $(\rho+\varepsilon)$-competitive algorithm $\mathcal{A}^*(\varepsilon)$ for MPS, for any $0 < \varepsilon \le 1$. The main idea is to repeatedly execute $\mathcal{A}$ for a set of guesses on the optimum makespan. The initial guesses are small and are increased whenever a guess turns out to be smaller than $\text{OPT}(\sigma)$. The increments are done in small steps so that, among the final guesses, there exists one that is upper bounded by approximately $(1+\varepsilon)\text{OPT}(\sigma)$. In the analysis of this scheme we will have to bound machine loads caused by scheduling "errors" made when guesses were too small. Unfortunately the execution of $\mathcal{A}$, given a guess $\gamma \ne \text{OPT}(\sigma)$, can lead to undefined algorithmic behavior. As we shall show, guesses $\gamma \ge \text{OPT}(\sigma)$ are not critical. However, guesses $\gamma < \text{OPT}(\sigma)$ have to be handled carefully.

So let $\mathcal{A} = \{A_k\}_{k\in\mathcal{K}}$ be a ρ-competitive algorithm for MPS$_{\text{opt}}$ that, given guess γ, is executed on a job sequence σ. Upon the arrival of a job J_t, an algorithm $A_k \in \mathcal{A}$ may *fail* because the scheduling rules of A_k do not specify a machine where to place J_t in the current schedule S_k. We define two further conditions when an algorithm A_k fails. The first one identifies situations where a makespan of $\rho\gamma$ is not preserved and hence ρ-competitiveness may not be guaranteed. More precisely, A_k would assign J_t to a machine M_j such that $\ell(j) + p_t > \rho\gamma$, where $\ell(j)$ denotes M_j's machine load before the assignment. The second condition identifies situations where γ is not consistent with lower bounds on the optimum makespan, i. e. γ is smaller than the average machine load or the processing time of J_t. Formally, an algorithm A_k *fails* if a job J_t, $1 \le t \le n$, has to be scheduled and one of the following conditions holds.

(i) A_k does not specify a machine where to place J_t in the current schedule S_k.
(ii) There holds $\ell(j) + p_t > \rho\gamma$, for the machine M_j to which A_k would assign J_t in S_k.
(iii) There holds $\gamma < \sum_{t'\le t} p_{t'}/m$ or $\gamma < p_t$.

We first show that guesses $\gamma \ge \text{OPT}(\sigma)$ are not problematic. If a ρ-competitive algorithm $\mathcal{A} = \{A_k\}_{k\in\mathcal{K}}$ for MPS$_{\text{opt}}$ is given a guess $\gamma \ge \text{OPT}(\sigma)$, then there exists an algorithm $A_k \in \mathcal{A}$ that does not fail during the processing of σ and generates a schedule whose makespan is at most $\rho\gamma$. This is shown by the next lemma.

Lemma 1. *Let $\mathcal{A} = \{A_k\}_{k\in\mathcal{K}}$ be a ρ-competitive algorithm for MPS$_{\text{opt}}$ that, given guess γ, is executed on a job sequence σ with $\gamma \ge \text{OPT}(\sigma)$. Then there exists an algorithm $A_k \in \mathcal{A}$ that does not fail during the processing of σ and generates a schedule whose makespan is at most $\rho\gamma$.*

Proof. Let S_{opt} be an optimal schedule for the job sequence $\sigma = J_1, \ldots, J_n$. Moreover, let $\ell(j)$ denote the load of machine M_j in S_{opt}, $1 \le j \le m$. For any j with $\ell(j) < \gamma$, define a job J'_j of processing time $p'_j = \gamma - \ell(j)$. Let σ' be the job sequence consisting of σ followed by the new jobs J'_j. These up to m jobs may be appended to σ in any order. Obviously $\text{OPT}(\sigma') = \gamma$. Hence when $\mathcal{A}$ using guess γ is executed on σ', there must exist an algorithm $A_{k^*} \in \mathcal{A}$ that generates a schedule with a makespan of at most $\rho\gamma$. Since σ is a prefix of σ', this algorithm A_{k^*} does not fail and generates a schedule with a makespan of at most $\rho\gamma$, when $\mathcal{A}$ given guess γ is executed on σ. □

Algorithm for MPS: We describe our algorithm $\mathcal{A}^*(\varepsilon, h)$ for MPS, where $0 < \varepsilon \le 1$ and $h \in \mathbb{N}$ may be chosen arbitrarily. The construction takes as input any algorithm $\mathcal{A} = \{A_k\}_{k \in \mathcal{K}}$ for MPS_{opt}. For a proper choice of h, $\mathcal{A}^*(\varepsilon, h)$ will be $(\rho + \varepsilon)$-competitive, provided that $\mathcal{A}$ is ρ-competitive.

At any time $\mathcal{A}^*(\varepsilon, h)$ works with h guesses $\gamma_1 < \ldots < \gamma_h$ on the optimum makespan for the incoming job sequence σ. These guesses may be adjusted during the processing of σ; the update procedure will be described in detail below. For each guess γ_i, $1 \le i \le h$, $\mathcal{A}^*(\varepsilon, h)$ executes $\mathcal{A}$. Hence $\mathcal{A}^*(\varepsilon, h)$ maintains a total of $h|\mathcal{K}|$ schedules, which can be partitioned into subsets $\mathcal{S}_1, \ldots, \mathcal{S}_h$. Subset $\mathcal{S}_i$ contains those schedules generated by $\mathcal{A}$ using γ_i, $1 \le i \le h$. Let $S_{ik} \in \mathcal{S}_i$ denote the schedule generated by A_k using γ_i.

A job sequence σ is processed as follows. Initially, upon the arrival of the first job J_1, the guesses are initialized as $\gamma_1 = p_1$ and $\gamma_i = (1 + \varepsilon)\gamma_{i-1}$, for $i = 2, \ldots, h$. Each job J_t, $1 \le t \le n$, is handled in the following way. Of course each such job is sequenced in every schedule S_{ik}, $1 \le i \le h$ and $1 \le k \le |\mathcal{K}|$. Algorithm $\mathcal{A}^*(\varepsilon, h)$ checks if A_k using γ_i fails when having to sequence J_t in S_{ik}. We remark that this check can be performed easily by just verifying if one of the conditions (i–iii) holds. If A_k using γ_i does not fail and has not failed since the last adjustment of γ_i, then in S_{ik} job J_t is assigned to the machine specified by A_k using γ_i. The initialization of a guess is also regarded as an adjustment. If A_k using γ_i does fail, then J_t and all future jobs are always assigned to a least loaded machine in S_{ik} until γ_i is adjusted the next time.

Suppose that after the sequencing of J_t all algorithms of $\mathcal{A} = \{A_k\}_{k \in \mathcal{K}}$ using a particular guess γ_i have failed since the last adjustment of this guess. Let i^* be the largest index i with this property. Then the guesses $\gamma_1, \ldots, \gamma_{i^*}$ are adjusted. Set $\gamma_1 = (1 + \varepsilon)\max\{\gamma_h, p_t, \sum_{1 \le t' \le t} p_{t'}/m\}$ and $\gamma_i = (1 + \varepsilon)\gamma_{i-1}$, for $i = 2, \ldots, i^*$. For any readjusted guess γ_i, $1 \le i \le i^*$, algorithm $\mathcal{A}$ using γ_i ignores all jobs $J_{t'}$ with $t' < t$ when processing future jobs of σ. Specifically, when making scheduling decisions and determining machine loads, algorithm A_k using γ_i ignores all job $J_{t'}$ with $t' < t$ in its schedule S_{ik}. These jobs are also ignored when $\mathcal{A}^*(\varepsilon, h)$ checks if A_k using guess γ_i fails on the arrival of a job. Furthermore, after the assignment of J_t, machines in S_{ik} machines are renumbered so that J_t is located on a machine it would occupy if it were the first job of an input sequence.

When guesses have been adjusted, they are renumbered, together with the corresponding schedule sets $\mathcal{S}_i$, such that again $\gamma_1 < \ldots < \gamma_h$. Hence at any time $\gamma_1 = \min_{1 \le i \le h} \gamma_i$ and $\gamma_i \ge (1 + \varepsilon)\gamma_{i-1}$, for $i = 2, \ldots, h$. We also observe that whenever a guess is adjusted, its value increases by a factor of at least $(1 + \varepsilon)^h$. A summary of $\mathcal{A}^*(\varepsilon, h)$ is given in Figure 1.

Algorithm $\mathcal{A}^*(\varepsilon, h)$

1. Set $\gamma_i = p_1(1+\varepsilon)^{i-1}$, for $i = 1, \ldots, h$.
2. At time t execute the following steps.
   (a) J_t is sequenced as follows in each S_{ik}. If A_k using γ_i fails or has failed since the last adjustment of γ_i, then assign J_t to a least loaded machine. Otherwise assign it to the machine specified by A_k, ignoring jobs that arrived before the last adjustment of γ_i.
   (b) If all algorithms $\{A_k\}_{k\in\mathcal{K}}$ for some γ_i have failed since the last readjustment of γ_i, then let i^* be the largest index with this property. Set $\gamma_i = (1+\varepsilon)^i \max\{\gamma_h, p_t, \sum_{t'\leq t} p_{t'}/m\}$, for $i = 1, \ldots, i^*$. Renumber the guesses such that $\gamma_1 < \ldots < \gamma_h$.

Fig. 1. The algorithm $\mathcal{A}^*(\varepsilon, h)$

We obtain the following theorem.

Theorem 1. *Let $\mathcal{A} = \{A_k\}_{k\in\mathcal{K}}$ be a ρ-competitive algorithm for MPS$_{\text{opt}}$. Then for any $0 < \varepsilon \leq 1$ and $h = \lceil \log(1+\frac{6\rho}{\varepsilon})/\log(1+\frac{\varepsilon}{3\rho})\rceil$, algorithm $\mathcal{A}^*(\varepsilon) = \mathcal{A}^*(\varepsilon/(3\rho), h)$ for MPS is $(\rho+\varepsilon)$-competitive and uses $h|\mathcal{K}|$ schedules.*

For the analysis of $\mathcal{A}^*(\varepsilon, h)$ we need the following lemma.

Lemma 2. *After $\mathcal{A}^*(\varepsilon, h)$ has processed a job sequence σ, there holds $\gamma_1 \leq (1+\varepsilon)\text{OPT}(\sigma)$.*

Proof. At any time $\mathcal{A}^*(\varepsilon, h)$ maintains h guesses. We can view these guesses as being stored in h variables. A variable is updated whenever its current guess is increased. Hence during the processing of σ a variable may take any position in the sorted sequence of guesses. We analyze the steps in which $\mathcal{A}^*(\varepsilon, h)$ adjusts guesses.

We first show that when $\mathcal{A}^*(\varepsilon, h)$ adjusts a guess γ, then $\gamma < \text{OPT}(\sigma)$. So suppose that after the arrival of a job J_t, $\mathcal{A}^*(\varepsilon, h)$ adjust guesses $\gamma_1, \ldots, \gamma_{i^*}$, where i^* is the largest index i such that all algorithms $\{A_k\}_{k\in\mathcal{K}}$ using γ_i have failed. We prove $\gamma_{i^*} < \text{OPT}(\sigma)$, which implies the desired statement because guesses are numbered in order of increasing value. Let t^*, with $t^* < t$, be the most recent time when the variable storing γ_{i^*} was updated last. If the variable has never been updated since its initialization, then let $t^* = 1$. All the algorithms $\{A_k\}_{k\in\mathcal{K}}$ using γ_{i^*} ignore the jobs having arrived before J_{t^*} when making scheduling decisions for $J_{t^*}, \ldots, J_t$. Let $\sigma^* = J_{t^*}, \ldots, J_t$. There holds, $\text{OPT}(\sigma^*) \leq \text{OPT}(\sigma)$. If $\gamma_{i^*} \geq \text{OPT}(\sigma)$ held true, then by Lemma 1 there would be an algorithm $A_{k^*} \in \{A_k\}_{k\in\mathcal{K}}$ that, using guess γ_{i^*}, does not fail when handling σ^*. This contradicts the fact that at time t all algorithms $\{A_k\}_{k\in\mathcal{K}}$ using γ_{i^*} fail or have failed since the arrival of J_{t^*}.

Let γ_1^e denote the value of the smallest guess when $\mathcal{A}^*(\varepsilon, h)$ has finished processing σ. We distinguish two cases depending on whether or not the variable storing γ_1^e has ever been updated since its initialization. If the variable has never been updated, then $\gamma_1^e = p_1(1+\varepsilon)^{i-1}$, for some $i \in \{1, \ldots, h\}$. If $i = 1$, there is nothing to show because

$p_1 \le \text{OPT}(\sigma)$. If $i > 1$, then the initial guess of value $\gamma_{i-1} = p_1(1+\varepsilon)^{i-2}$ must have been adjusted. This implies, as shown above, $\gamma_{i-1} < \text{OPT}(\sigma)$ and the lemma follows because $\gamma_1^e = (1+\varepsilon)\gamma_{i-1}$.

In the remainder of the proof we assume that the variable g storing γ_1^e has been updated. Consider the last update of g before the end of σ and suppose that it took place on the arrival of job J_{t^*}. First assume that g stores the smallest guess, among the h guesses, before the update. Then $\gamma_1^e = (1+\varepsilon)\max\{\gamma^*, p_{t^*}, \sum_{1\le t'\le t^*} p_{t'}/m\}$, where γ^* is the largest guess before the update. If γ^* is also adjusted on the arrival of J_{t^*}, then we are done because, as shown above, $\gamma^* < \text{OPT}(\sigma)$ and thus $\max\{\gamma^*, p_{t^*}, \sum_{1\le t'\le t^*} p_{t'}/m\} \le \text{OPT}(\sigma)$. If γ^* is not adjusted on the arrival of J_{t^*}, then γ_1^e is the smallest guess greater than γ^* after the update. By the end of σ guess γ^* must be adjusted since otherwise γ_1^e cannot become the smallest guess. Again $\gamma^* < \text{OPT}(\sigma)$ and we are done.

Finally assume that before the update g does not store the smallest guess. Let g' be the variable that stores the largest guess smaller than that in g. After the update there holds $\gamma_1^e = (1+\varepsilon)\gamma$, where γ is the guess stored in g' after the update. Until the end of σ, γ must be adjusted again since otherwise γ_1^e cannot become the smallest guess. Again $\gamma < \text{OPT}(\sigma)$ and hence $\gamma_1^e < (1+\varepsilon)\text{OPT}(\sigma)$. □

Proof (of Theorem 1). Throughout the proof let $h = \lceil \log(1+\frac{6\rho}{\varepsilon})/\log(1+\frac{\varepsilon}{3\rho})\rceil$ and $\mathcal{A}^*(\varepsilon) = \mathcal{A}^*(\varepsilon/(3\rho), h)$. Consider an arbitrary job sequence and let γ_1 be the smallest of the h guesses maintained by $\mathcal{A}^*(\varepsilon)$ at the end of σ. Let $\mathcal{S}_1$ be the set of schedules associated with γ_1, i. e. $\mathcal{S}_1$ was generated by $\mathcal{A} = \{A_k\}_{k\in\mathcal{K}}$ using a series of guesses ending with γ_1. Let $\gamma(0) < \ldots < \gamma(s)$, with $s \ge 0$, be this series and g be the variable that stored these guesses. Here $\gamma(0)$ is one of the initial guesses and $\gamma(s) = \gamma_1$.

A first observation is that at the end of σ there exists an algorithm $A_{k^*} \in \{A_k\}_{k\in\mathcal{K}}$ that using γ_1 has not failed. This holds true if g was set to $\gamma_1 = \gamma(s)$ upon the arrival of a job J_t with $t < n$ because the failure of all algorithms $\{A_k\}_{k\in\mathcal{K}}$ using γ_1 would have caused an adjustment of γ_1. This also holds true if g was set to γ_1 upon the arrival of J_n because in this case none of the algorithms $\{A_k\}_{k\in\mathcal{K}}$ using γ_1 has failed at the end of σ. So let $A_{k^*} \in \{A_k\}_{k\in\mathcal{K}}$ be an algorithm that using γ_1 has not failed and let S_{1k^*} be the associated schedule. We prove that the load of every machine in S_{1k^*} is upper bounded by $(\rho+\varepsilon)\text{OPT}(\sigma)$. This establishes the theorem.

Let $t_0 = 1$. If the variable g was updated during the processing of σ, then let $t_1, \ldots, t_s$ be these points in time, i. e. the arrival of J_{t_i} caused an update of g and the variable was set to $\gamma(i)$, $1 \le i \le s$. For any machine M_j, $1 \le j \le m$, in S_{1k^*} let $\ell(j)$ denote its final load at the end of σ. Moreover, let $\ell_{t_i}(j)$ denote its load due to jobs J_t with $t \ge t_i$, for $i = 0, \ldots, s$. Obviously

$$\ell(j) = \ell_{t_s}(j) + \sum_{i=0}^{s-1} (\ell_{t_i}(j) - \ell_{t_{i+1}}(j)) . \tag{1}$$

We first show that $\ell_{t_s}(j) \le \rho\gamma_1$. Immediately after J_{t_s} has been scheduled M_j's load consisting of jobs $J_{t'}$ with $t' \ge t_s$ is at most p_{t_s}. Since g was set to $\gamma(s) = \gamma_1$ on the arrival of J_{t_s}, the guess adjustment rule ensures $p_{t_s} \le \gamma_1$. Until the end of σ algorithm A_{k^*} using γ_1 does not fail and hence condition (ii) specifying the failure of algorithms implies that the assignment of each further job does not create a machine load greater than $\rho\gamma_1$ in S_{1k^*}.

We next show $\ell_{t_i}(j) - \ell_{t_{i+1}}(j) \le \max\{\rho, 2\}\gamma(i)$, for each $i = 0, \ldots, s-1$. The latter difference is the load on machine M_j caused by jobs of the subsequence $J_{t_i}, \ldots, J_{t_{i+1}-1}$. Hence it suffices to show that after the assignment of any J_t, with $t_i \le t < t_{i+1}$, M_j's load due to jobs $J_{t'}$, with $t' \ge t_i$, is at most $\max\{\rho, 2\}\gamma(i)$. After the assignment of J_{t_i} M_j's respective load $\ell_{t_i}(j)$ is at most p_{t_i} and this value is upper bounded by $\gamma(i)$ as ensured by the guess adjustment rule. At times $t > t_i$, while A_{k^*} using $\gamma(i)$ has not failed, M_j's load due to jobs $J_{t'}$ with $t' \ge t_i$ does not exceed $\rho\gamma(i)$ as ensured by condition (ii) specifying the failure of algorithms. Finally consider a time t, $t_i < t < t_{i+1}$, at which A_{k^*} fails or has failed. The incoming job J_t is assigned to a least loaded machine. Hence if J_t is placed on M_j, then the resulting machine load due to jobs $J_{t'}$ with $t' \ge t_i$ is upper bounded by $\sum_{t_i \le t' < t} p_{t'}/m + p_t \le \sum_{1 \le t' \le t} p_{t'}/m + p_t$. Observe that after the arrival of J_t there exists an algorithm $A_k \in \mathcal{A}$ that using $\gamma(i)$ has not yet failed, since otherwise $\gamma(i)$ would be adjusted before time t_{i+1}. Condition (iii) defining the failure of algorithms ensures that $\sum_{1 \le t' \le t} p_{t'}/m \le \gamma(i)$ and $p_t \le \gamma(i)$. We obtain that M_j's machine load is at most $2\gamma(i)$.

We conclude that (1) is upper bounded by

$$\rho\gamma_1 + \sum_{i=0}^{s-1} \max\{\rho, 2\}\gamma(i). \tag{2}$$

By Lemma 2, $\gamma_1 = \gamma(s) \le (1 + \varepsilon/(3\rho))\mathrm{OPT}(\sigma)$. At the end of the description of $\mathcal{A}^*(\varepsilon, h)$ we observed that whenever a guess is adjusted it increases by a factor of at least $(1+\varepsilon)^h$. Hence $\gamma(i) \ge (1+\varepsilon/(3\rho))^h\gamma(i-1)$. It follows that $\gamma(i) \le \frac{\gamma(s)}{(1+(\varepsilon/3\rho))^{(s-i)\cdot h}}$, for every $0 \le i \le s$. Hence (2) is upper bounded by

$$\begin{aligned}
&\rho(1 + \frac{\varepsilon}{3\rho})\mathrm{OPT}(\sigma) + \sum_{i=0}^{s-1} \frac{\max\{\rho, 2\}\gamma(s)}{(1+\varepsilon/(3\rho))^{h\cdot(s-i)}} \\
&\le \rho(1 + \frac{\varepsilon}{3\rho})\mathrm{OPT}(\sigma) + \rho(1 + \frac{\varepsilon}{3\rho})\mathrm{OPT}(\sigma) \sum_{i=0}^{s-1} \frac{2}{(1+\varepsilon/(3\rho))^{h\cdot(s-i)}} && (3) \\
&\le \rho(1 + \frac{\varepsilon}{3\rho})\mathrm{OPT}(\sigma) \left(1 + \sum_{i=1}^{\infty} \frac{2}{(1+\varepsilon/(3\rho))^{h\cdot i}}\right) \\
&= \rho(1 + \frac{\varepsilon}{3\rho})\mathrm{OPT}(\sigma) \left(1 + \frac{2}{(1+\varepsilon/(3\rho))^{h} - 1}\right) && (4) \\
&\le \rho(1 + \frac{\varepsilon}{3\rho})^2\mathrm{OPT}(\sigma) \ \le \ \rho(1 + \frac{\varepsilon}{\rho})\mathrm{OPT}(\sigma) = (\rho + \varepsilon)\mathrm{OPT}(\sigma). && (5)
\end{aligned}$$

Here (3) uses the fact that $\max\{\rho, 2\} \le 2\rho$ and, as mentioned above, is a consequence of Lemma 2. Line (4) follows from the Geometric Series and, finally, (5) is by the choice of h and the assumption $0 < \varepsilon \le 1$. □

3 A $(1+\varepsilon)$-competitive algorithm for $\mathrm{MPS_{opt}}$

We present an algorithm $\mathcal{A}_1(\varepsilon)$ for $\mathrm{MPS_{opt}}$ that attains a competitive ratio of $1+\varepsilon$, for any $\varepsilon > 0$. The number of parallel schedules will be $(\lfloor 2m/\varepsilon \rfloor + 1)^{\lceil \log(2/\varepsilon)/\log(1+\varepsilon/2) \rceil}$.

The algorithms will yield a $(1+\varepsilon)$-competitive strategy for MPS and, furthermore, will be useful in the next section where we develop a $(4/3+\varepsilon)$-competitive algorithm for $\mathrm{MPS}_{\mathrm{opt}}$. There $\mathcal{A}_1(\varepsilon)$ will be used as subroutine for a small, constant number of m.

Description of $\mathcal{A}_1(\varepsilon)$: Let $\varepsilon > 0$ be arbitrary. Recall that in $\mathrm{MPS}_{\mathrm{opt}}$ the optimum makespan $\mathrm{OPT}(\sigma)$ for the incoming job sequence σ is initially known. Assume without loss of generality that $\mathrm{OPT}(\sigma) = 1$. Then all job processing times are in $(0,1]$. Set $\varepsilon' = \varepsilon/2$. First we partition the range of possible job processing times into intervals $I_0, \ldots, I_l$ such, within each interval I_i with $i \geq 1$, the values differ by a factor of at most $1+\varepsilon'$. Such a partitioning is standard and has been used e. g. in the PTAS for offline makespan minimization [23]. Let $l = \lceil \log(1/\varepsilon')/\log(1+\varepsilon') \rceil$. Set $I_0 = (0, \varepsilon']$ and $I_i = ((1+\varepsilon')^{i-1}\varepsilon', (1+\varepsilon')^i\varepsilon']$, for $i = 1, \ldots, l$. Obviously $I_0 \cup \ldots \cup I_l = (0, (1+\varepsilon')^l\varepsilon']$ and $(0,1] \subseteq (0, (1+\varepsilon')^l\varepsilon']$. A job is *small* if its processing time is at most ε' and hence contained in I_0; otherwise the job is *large*.

Each job sequence σ with $\mathrm{OPT}(\sigma) = 1$ contains at most $\lfloor m/\varepsilon' \rfloor$ large jobs. For each possible distribution of large jobs over the processing time intervals $I_1, \ldots, I_l$, algorithm $\mathcal{A}_1(\varepsilon)$ prepares one algorithm/schedule. Let $V = \{(v_1, \ldots, v_l) \in \mathbb{N}_0^l \mid v_i \leq \lfloor m/\varepsilon' \rfloor\}$. There holds $|V| = (\lfloor m/\varepsilon' \rfloor + 1)^l$. Let $\mathcal{A}_1(\varepsilon) = \{A_v\}_{v \in V}$. For any vector $v = (v_1, \ldots, v_n) \in V$, algorithm A_v works as follows. It assumes that the incoming job sequence σ contains exactly v_i jobs with a processing time in I_i, for $i = 1, \ldots, l$. Moreover, it pessimistically assumes that each processing time in I_i takes the largest possible value $(1+\varepsilon')^i\varepsilon'$. Hence, initially A_v computes an optimal schedule S_v^* for a job sequence consisting of v_i jobs with a processing time of $(1+\varepsilon')^i\varepsilon'$, for $i = 1, \ldots, l$. Small jobs are ignored. Since running time is not an issue in the design of online algorithms, such a schedule S_v^* can be computed exactly. Alternatively, an $(1+\varepsilon')$-approximation to the optimal schedule can be computed using the PTAS by Hochbaum and Shmoys [23]. Let $n_i^*(j)$ denote the number of jobs with a processing time of $(1+\varepsilon')^i\varepsilon' \in I_i$ assigned to machine M_j in S_v^*, where $1 \leq i \leq l$ and $1 \leq j \leq m$. Moreover, let $\ell^*(j) = \sum_{i=1}^{l} n_i^*(j)(1+\varepsilon')^i\varepsilon'$ be the load on machine M_j in S_v^*, $1 \leq j \leq m$.

When processing the actual job sequence σ and constructing a real schedule S_v, A_v uses S_v^* as a guideline to make scheduling decisions. At any time during the scheduling process, let $n_i(j)$ be the number of jobs with a processing time in I_i that have already been assigned to machine M_j in S_v, where again $1 \leq i \leq l$ and $1 \leq j \leq m$. Each incoming job J_t, $1 \leq t \leq n$, is handled as follows. If J_t is large, then let I_i with $1 \leq i \leq l$ be the interval such that $p_t \in I_i$. Algorithm A_v checks if there is a machine M_j such that $n_i^*(j) - n_i(j) > 0$, i. e. there is a machine that can still accept a job with a processing time in I_i as suggested by the optimal schedule S_v^*. If such a machine M_j exists, then J_t is assigned to it; otherwise J_t is scheduled on an arbitrary machine. If J_t is small, then J_t is assigned to a machine M_j with the smallest current value $\ell^*(j) + \ell_s(j)$. Here $\ell_s(j)$ denotes the current load on machine M_j caused by small jobs in S_v. A summary of $\mathcal{A}_1(\varepsilon)$ is given in Figure 2. Subsequently we show Theorem 2.

Theorem 2. *For any $\varepsilon > 0$, $\mathcal{A}_1(\varepsilon)$ is $(1+\varepsilon)$-competitive and uses at most $(\lfloor 2m/\varepsilon \rfloor + 1)^{\lceil \log(2/\varepsilon)/\log(1+\varepsilon/2) \rceil}$ schedules.*

Proof. The bound on the number of schedules simply follows from the fact that $\mathcal{A}_1(\varepsilon)$ maintains $|V| = (\lfloor m/\varepsilon' \rfloor + 1)^l$ schedules where $\varepsilon' = \varepsilon/2$ and $l = \lceil \log(1/\varepsilon')/\log(1+\varepsilon') \rceil$.

Algorithm $\mathcal{A}_1(\varepsilon)$

1. $\mathcal{A}_1(\varepsilon) = \{A_v\}_{v \in V}$, where $V = \{(v_1, \ldots, v_l) \in \mathbb{N}_0^l \mid v_i \le \lfloor m/\varepsilon' \rfloor\}$ with $\varepsilon' = \varepsilon/2$ and $l = \lceil \log(1/\varepsilon')/\log(1+\varepsilon') \rceil$.
2. A_v works as follows.
   (a) Compute optimal schedule S_v^* for input consisting of v_i jobs of processing time $(1+\varepsilon')^i\varepsilon'$, $1 \le i \le l$.
   (b) In S_v each J_t is sequenced in the following way.
   If $p_t > \varepsilon'$, then determine I_i such that $p_t \in I_i$. If $\exists\, M_j$ with $n_i^*(j) - n_i(j) > 0$, then assign J_t to it; otherwise assign J_t to an arbitrary machine.
   If $p_t \le \varepsilon'$, then assign J_t to M_j with the smallest value $\ell^*(j) + \ell_s(j)$.

Fig. 2. The algorithm $\mathcal{A}_1(\varepsilon)$

Let σ be an arbitrary job sequence and let v_i be the number of jobs with a processing time in I_i, for $i = 1, \ldots, l$. Since any v_i is upper bounded by $\lfloor m/\varepsilon' \rfloor$, the resulting vector $v = (v_1, \ldots, v_l)$ is in V. For this vector v, consider the associated algorithm A_v. We prove that when A_v has finished processing σ, the resulting schedule S_v has a makespan of at most $(1+\varepsilon) = (1+\varepsilon)\text{OPT}(\sigma)$. Recall again that we assume without loss of generality that $\text{OPT}(\sigma) = 1$.

We analyze the steps in which A_v assigns jobs J_t, $1 \le t \le n$, to machines in S_v. If J_t is large with $p_t \in I_i$, $1 \le i \le l$, then there must exist a machine M_j in the current schedule S_v such that $n_i^*(j) - n_i(j) > 0$. Algorithm A_v will assign J_t to such a machine. Hence after the processing of σ, for any M_j in S_v, the total load caused by large jobs is upper bounded by $\ell^*(j)$. We next argue that this value is at most $(1+\varepsilon')\text{OPT}(\sigma)$. Consider an optimal schedule S_{opt} for σ. Modify this schedule by (a) deleting all small jobs and (b) rounding each job processing time in I_i to $(1+\varepsilon')^i\varepsilon'$, for $i = 1, \ldots, l$. The resulting schedule schedule S'_{opt} has a makespan of at most $(1+\varepsilon')\text{OPT}(\sigma)$. Furthermore S'_{opt} is a schedule for an input sequence consisting of v_i jobs of processing time $(1+\varepsilon')^i\varepsilon'$. Since S_v^* is an optimal schedule for this input, each machine load $\ell^*(j)$ is upper bounded by $(1+\varepsilon')\text{OPT}(\sigma)$.

We finally show that when A_v has to sequence a small job J_t, then there is a machine M_j such that $\ell^*(j) + \ell_s(j)$ is upper bounded by $(1+\varepsilon')\text{OPT}(\sigma)$. This implies that the assignment of J_t causes a machine load of at most $(1+\varepsilon')\text{OPT}(\sigma) + p_t \le (1 + 2\varepsilon')\text{OPT}(\sigma) = (1+\varepsilon)\text{OPT}(\sigma)$ in the final schedule S_v.

So suppose that upon the arrival of a small job J_t there holds $\ell^*(j) + \ell_s(j) > (1+\varepsilon')\text{OPT}(\sigma)$ for all machines M_j, $1 \le j \le m$. Recall that $\ell_s(j)$ is the load on machine M_j caused by small jobs in the current schedule S_v. Note that $\sum_{j=1}^m \ell^*(j)$ is the total processing time of large jobs in σ if processing times in I_i are rounded up to $(1+\varepsilon')^i\varepsilon'$, for $i = 1, \ldots, l$. Hence $1/(1+\varepsilon)\sum_{j=1}^m \ell^*(j)$ is a lower bound on the total processing time of large jobs in σ. It follows that the total processing time of all jobs in σ is at least $1/(1+\varepsilon')\sum_{j=1}^m \ell^*(j) + \sum_{j=1}^m \ell_s(j) + p_t \ge 1/(1+\varepsilon')\sum_{j=1}^m (\ell^*(j) + \ell_s(j)) + p_t$. The assumption that $\ell^*(j) + \ell_s(j) > (1+\varepsilon')\text{OPT}(\sigma)$ holds for all machines M_j implies that the total processing time of jobs in σ is at least $m \cdot \text{OPT}(\sigma) + p_t > m \cdot \text{OPT}(\sigma)$, which contradicts the fact that $\text{OPT}(\sigma)$ is the optimum makespan. $\square$

4 A $(4/3+\varepsilon)$-competitive algorithm for $\mathrm{MPS_{opt}}$

We develop an algorithm $\mathcal{A}_2(\varepsilon)$ for $\mathrm{MPS_{opt}}$ that is $(4/3+\varepsilon)$-competitive, for any $0 < \varepsilon \leq 1$, if the number m of machines is not too small. We then combine $\mathcal{A}_2(\varepsilon)$ with $\mathcal{A}_1(\varepsilon)$, presented in the last section, and derive a strategy $\mathcal{A}_3(\varepsilon)$ that is $(4/3+\varepsilon)$-competitive, for arbitrary m. The number of required schedules is $1/\varepsilon^{O(\log(1/\varepsilon))}$, which is a constant independent of n and m. We firstly present a description of the algorithm; the corresponding analysis is given thereafter.

Before describing $\mathcal{A}_2(\varepsilon)$ in detail, we explain the main ideas of the algorithm. One concept is identical to that used by $\mathcal{A}_1(\varepsilon)$: Partition the range of possible job processing times into intervals or *job classes* and consider distributions of jobs over these classes. However, in order to achieve a constant number of schedules we have to refine this scheme and incorporate new ideas. First, the job classes have to be chosen properly so as to allow a compact packing of jobs on the machines. An important, new aspect in the construction of $\mathcal{A}_2(\varepsilon)$ is that we will not consider the entire set V of tuples specifying how large jobs of an input sequence σ are distributed over the job classes. Instead we will define a suitable sparsification V' of V. Each $v \in V'$ represents an estimate or guess on the number of large jobs arising in σ. More specifically, if $v = (v_1, \ldots, v_l)$, then it is assumed that σ contains at least v_i jobs with a processing time of job class i.

Obviously, the job sequence σ may contain more large jobs, the exact number of which is unknown. Furthermore, it is unknown which portion of the total processing time of σ will arrive as small jobs. In order to cope with these uncertainties $\mathcal{A}_2(\varepsilon)$ has to construct robust schedules. To this end the number of machines is partitioned into two sets $\mathcal{M}_c$ and $\mathcal{M}_r$. For the machines of $\mathcal{M}_c$, the algorithm initially determines a good assignment or *configuration* assuming that v_i jobs of job class i will arrive. The machines of $\mathcal{M}_r$ are reserve machines and will be assigned additional large jobs as they arise in σ. Small jobs will always be placed on machines in $\mathcal{M}_c$. The initial configuration determined for these machines has the property that, no matter how many small jobs arrive, a machine load never exceeds $4/3+\varepsilon$ times the optimum makespan.

We proceed to describe $\mathcal{A}_2(\varepsilon)$ in detail. Let $0 < \varepsilon \leq 1$. Moreover, set $\varepsilon' = \varepsilon/8$. Again we assume without loss of generality that, for an incoming job sequence, there holds $\mathrm{OPT}(\sigma) = 1$. Hence the processing time of any job is upper bounded by 1.

Job classes: A job J_t, $1 \leq t \leq n$, is *small* if $p_t \leq 1/3+2\varepsilon'$; otherwise J_t is *large*. We divide the range of possible job processing times into job classes. Let $I_s = (0, 1/3+2\varepsilon']$ be the interval containing the processing times of small jobs. Let $\lambda = \lceil \log(\frac{3}{8} + \frac{1}{48\varepsilon'}) \rceil$ and $l = \lambda + 2$, where the logarithm is taken to base 2. For $i = 1, \ldots, l$, let

$$a_i = \max\{\tfrac{1}{3} - 2\varepsilon' + (\tfrac{1}{12} + \tfrac{3}{2}\varepsilon')\tfrac{1}{2^{\lambda+1-i}}, \tfrac{1}{3} + 2\varepsilon'\} \quad \text{and} \quad b_i = \tfrac{1}{3} - 2\varepsilon' + (\tfrac{1}{12} + \tfrac{3}{2}\varepsilon')\tfrac{1}{2^{\lambda-i}}.$$

It is easy to verify that $a_1 = 1/3 + 2\varepsilon'$ and $a_i < b_i$, for $i = 1, \ldots, l$. Furthermore $b_{l-1} = 1/2 + \varepsilon'$ and $b_l = 2/3 + 4\varepsilon'$. For $i = 1, \ldots, l$ define $I_i = (a_i, b_i]$. There holds $\bigcup_{1 \leq i \leq l} I_i = (1/3+2\varepsilon', 2/3+4\varepsilon']$. Moreover, for $i = 1, \ldots, l-1$, let $I_{l+i} = (2a_i, 2b_i]$. Intuitively, I_{l+i} contains the processing times that are twice as large as those in I_i, $1 \leq i \leq l-1$. There holds $\bigcup_{1 \leq i \leq l-1} I_{l+i} = (2/3+4\varepsilon', 1+2\varepsilon']$. Hence $I_s \cup I_1 \cup \ldots \cup I_{2l-1} = (0, 1+2\varepsilon']$. In the following I_i represents *job class* i, for $i = 1, \ldots, 2l-1$. We say that J_t is a *class-i job* if $p_t \in I_i$, where $1 \leq i \leq 2l-1$.

Definition of target configurations: As mentioned above, for any incoming job sequence σ, $\mathcal{A}_2(\varepsilon)$ works with estimates on the number of class-i jobs arising in σ, $1 \le i \le 2l-1$. For each estimate, the algorithm initially determines a virtual schedule or *target configuration* on a subset of the machines, assuming that the estimated set of large jobs will indeed arrive. Hence we partition the m machines into two sets $\mathcal{M}_c$ and $\mathcal{M}_r$. Let $\mu = \lceil (1+\varepsilon')/(1+2\varepsilon') \cdot m \rceil$. Moreover, let $\mathcal{M}_c = \{M_1, \dots, M_\mu\}$ and $\mathcal{M}_r = \{M_{\mu+1}, \dots, M_m\}$. Set $\mathcal{M}_c$ contains the machines for which a target configuration will be computed; $\mathcal{M}_r$ contains the reserve machines. The proportion of $|\mathcal{M}_r|$ to $|\mathcal{M}_c|$ is roughly $1 : 1 + 1/\varepsilon'$.

A target configuration has the important property that any machine $M_j \in \mathcal{M}_c$ contains large jobs of only one job class i, $1 \le i \le 2l-1$. Therefore, a target configuration is properly defined by a vector $c = (c_1, \dots, c_\mu) \in \{0, \dots, 2l-1\}^\mu$. If $c_j = 0$, then M_j does not contain any large jobs in the target configuration, $1 \le j \le \mu$. If $c_j = i$, where $i \in \{1, \dots, 2l-1\}$, then M_j contains class-i jobs, $1 \le j \le \mu$. The vector c implicitly also specifies how many large jobs reside on a machine. If $c_j = i$ with $1 \le i \le l$, then M_j contains two class-i jobs. Note that, for general $i \in \{1, \dots, l\}$, a third job cannot be placed on the machine without exceeding a load bound of $4/3 + \varepsilon$. If $c_j = i$ with $l+1 \le i \le 2l-1$, then M_j contains one class-i job. Again, the assignment of a second job is not feasible in general. Given a configuration c, M_j is referred to as a *class-i machine* if $c_j = i$, where $1 \le j \le \mu$ and $1 \le i \le 2l-1$.

With the above interpretation of target configurations, each vector $c = (c_1, \dots, c_\mu)$ encodes inputs containing $2|\{c_j \in \{c_1, \dots c_\mu\} : c_j = i\}|$ class-i jobs, for $i = 1, \dots, l$, as well as $|\{c_j \in \{c_1, \dots c_\mu\} : c_j = i\}|$ class-i jobs, for $i = l+1, \dots, 2l-1$. Hence, for an incoming job sequence, instead of considering estimates on the number of class-i jobs, for any $1 \le i \le 2l-1$, we can equivalently consider target configurations. Unfortunately, it will not be possible to work with all target configurations $c \in \{0, \dots, 2l-1\}^\mu$ since the resulting number of schedules to be constructed would be $(2l)^\mu = (\log(1/\varepsilon))^{\Omega(m)}$. Therefore, we will work with a suitable sparsification of the set of all configurations.

Sparsification of the set of target configurations: Let $\kappa = \lceil 2(2+1/\varepsilon')(2l-1) \rceil$ and $U = \{0. \dots, \kappa\}^{2l-1}$. We will show that $\kappa \lfloor (m-\mu)/(2l-1) \rfloor \ge m$ if m is not too small (see Lemma 4). This property in turn will ensure that any job sequence σ can be mapped to a $u \in U$. For any vector $u = (u_1, \dots, u_{2l-1}) \in U$, we define a target configuration $c(u)$ that contains $u_i \lfloor (m-\mu)/(2l-1) \rfloor$ class-i machines, for $i = 1, \dots, 2l-1$, provided that $\sum_{i=1}^{2l-1} u_i \lfloor (m-\mu)/(2l-1) \rfloor$ does not exceed μ. More specifically, for any $u = (u_1, \dots, u_{2l-1}) \in U$, let $\pi_0 = 0$ and $\pi_i = \sum_{j=1}^{i} u_j \lfloor (m-\mu)/(2l-1) \rfloor$, be the partial sums of the first i entries of u, multiplied by $\lfloor (m-\mu)/(2l-1) \rfloor$, for $i = 1, \dots, 2l-1$. Let $\mu' = \pi_{2l-1}$. First construct a vector $c'(u) = (c'_1, \dots, c'_{\mu'})$ of length μ' that contains exactly $u_i \lfloor (m-\mu)/(2l-1) \rfloor$ class-i machines. That is, for $i = 1, \dots, 2l-1$, let $c'_j = i$ for $j = \pi_{i-1}+1, \dots, \pi_i$. We now truncate or extend $c'(u)$ to obtain a vector of length μ. If $\mu' \ge \mu$, then $c(u)$ is the vector consisting of the first μ entries of $c'(u)$. If $\mu' < \mu$, then $c(u) = (c'_1, \dots, c'_{\mu'}, 0, \dots, 0)$, i. e. the last $\mu - \mu'$ entries are set to 0. Let $C = \{c(u) \mid u \in U\}$ be the set of all target configurations constructed from vectors $u \in U$.

The algorithm family: Let $\mathcal{A}_2(\varepsilon) = \{A_c\}_{c \in C}$. For any $c \in C$, algorithm A_c works as follows. Initially, prior to the arrival of any job of σ, A_c determines the target configuration specified by $c = (c_1, \dots, c_\mu)$ and uses this virtual schedule for the machines of

$\mathcal{M}_c$ to make scheduling decisions. Consider a machine $M_j \in \mathcal{M}_c$ and suppose $c_j > 0$, i. e. M_j is a class-i machine for some $i \geq 1$. Let $\ell^-(j)$ and $\ell^+(j)$ be the targeted minimal and maximal loads caused by large jobs on M_j, according to the target configuration. More precisely, if $i \in \{1, \ldots, l\}$, then $\ell^-(j) = 2a_i$ and $\ell^+(j) = 2b_i$. Recall that in a target configuration a class-i machine contains two class-i jobs if $1 \leq i \leq l$. If $i \in \{l+1, \ldots, 2l-1\}$ and hence $i = l + i'$ for some $i' \in \{1, \ldots, l-1\}$, then $\ell^-(j) = 2a_{i'}$ and $\ell^+(j) = 2b_{i'}$. If $M_j \in \mathcal{M}_c$ is a machine with $c_j = 0$, then $\ell^-(j) = \ell^+(j) = 0$. While the job sequence σ is processed, a machine $M_j \in \mathcal{M}_c$ may or may not be *admissible*. Again assume that M_j is a class-i machine with $i \geq 1$. If $i \in \{1, \ldots, l\}$, then at any time during the scheduling process M_j is admissible if it has received less than two class-i jobs so far. Analogously, if $i \in \{l+1, \ldots, 2l-1\}$, then M_j is admissible if it has received no class-i job so far. Finally, at any time during the scheduling process, let $\ell(j)$ be the current load of machine M_j and let $\ell_s(j)$ be the load due to small jobs, $1 \leq j \leq m$.

Algorithm A_c schedules each incoming job J_t, $1 \leq t \leq n$, in the following way. First assume that J_t is a large job and, in particular, a class-i job, $1 \leq i \leq 2l-1$. The algorithm checks if there is a class-i machine in $\mathcal{M}_c$ that is admissible. If so, J_t is assigned to such a machine. If there is no admissible class-i machine available, then J_t is placed on a machine in $\mathcal{M}_r$. There jobs are scheduled according to the *Best-Fit* policy. More specifically, A_c checks if there exists a machine $M_j \in \mathcal{M}_r$ such that $\ell(j) + p_t \leq 4/3 + \varepsilon$. If this is the case, then J_t is assigned to such a machine with the largest current load $\ell(j)$. If no such machine exists, J_t is assigned to an arbitrary machine in $\mathcal{M}_r$. Next assume that J_t is small. The job is a assigned to a machine in $\mathcal{M}_c$, where preference is given to machines that have already received small jobs. Algorithm A_c checks if there is an $M_j \in \mathcal{M}_c$ with $\ell_s(j) > 0$ such that $\ell^+(j) + \ell_s(j) + p_t \leq 4/3 + \varepsilon$. If this is the case, then J_t is assigned to any such machine. Otherwise A_c considers the machines of $\mathcal{M}_c$ which have not yet received any small jobs. If there exists an $M_j \in \mathcal{M}_c$ with $\ell_s(j) = 0$ such that $\ell^+(j) + p_t \leq 4/3 + \varepsilon$, then among these machines J_t is assigned to one having the smallest targeted load $\ell^-(j)$. If again no such machine exists, J_t is assigned to an arbitrary machine in $\mathcal{M}_c$. A summary of $\mathcal{A}_2(\varepsilon)$, which focuses on the job assignment rules, is given in Figure 3. We obtain the following result.

Theorem 3. *$\mathcal{A}_2(\varepsilon)$ is $(4/3 + \varepsilon)$-competitive, for any $0 < \varepsilon \leq 1$ and $m \geq 2l/(\varepsilon')^2$. The algorithm uses $1/\varepsilon^{O(\log(1/\varepsilon))}$ schedules.*

$\mathcal{A}_2(\varepsilon)$ is $(4/3 + \varepsilon)$-competitive if, for the chosen ε, the number of machines is at least $2l/(\varepsilon')^2$. If the number of machines is smaller, we can simply apply algorithm $\mathcal{A}_1(\varepsilon)$ with an accuracy of $\varepsilon_0 = 1/3$. Let $\mathcal{A}_3(\varepsilon)$ be the following combined algorithm. If for the chosen ε, $m < 2l/(\varepsilon')^2$, execute $\mathcal{A}_1(1/3)$. Otherwise execute $\mathcal{A}_2(\varepsilon)$.

Corollary 1. *$\mathcal{A}_3(\varepsilon)$ is $(4/3+\varepsilon)$-competitive, for any $0 < \varepsilon \leq 1$, and uses $1/\varepsilon^{O(\log(1/\varepsilon))}$ schedules.*

Proof. If $\mathcal{A}_1(1/3)$ is executed for a machine number $m < 2l/(\varepsilon')^2$, then by Theorem 2 the number of schedules is $(\log(1/\varepsilon)/\varepsilon^3)^{O(1)}$, which is $1/\varepsilon^{O(1)}$. □

In the remainder of this section we prove Theorem 3. The stated number of schedules follows from the fact that $\mathcal{A}_2(\varepsilon)$ consists of $|C| = (\kappa + 1)^{2l-1}$ algorithms. Recall

Algorithm $\mathcal{A}_2(\varepsilon)$

1. $\mathcal{A}_2(\varepsilon) = \{A_c\}_{c\in C}$, where $C = \{c(u) \mid u \in U\}$
   $U = \{0, \dots, \kappa\}^{2l-1}$, where $\kappa = \lceil 2(2 + 1/\varepsilon')(2l - 1)\rceil$, $l = \lceil \log(\frac{3}{8} + \frac{1}{48\varepsilon'})\rceil + 2$ and $\varepsilon' = \varepsilon/8$
   $\mu = \lceil (1 + \varepsilon')/(1 + 2\varepsilon') \cdot m\rceil$
2. A_c works as follows.
   (a) Determine target configuration specified by $c = (c_1, \dots, c_\mu)$.
   (b) Each J_t is sequenced as follows.
   J_t is large: Let J_t be a class-i job. If there is an admissible class-i machine in $\mathcal{M}_c$, assign J_t to it. Otherwise check if $\exists\, M_j \in \mathcal{M}_r$ such that $\ell(j) + p_t \le 4/3 + \varepsilon$. If so, assign J_t to such an M_j with the highest $\ell(j)$; otherwise place J_t on an arbitrary $M_j \in \mathcal{M}_r$.
   J_t is small: If $\exists\, M_j \in \mathcal{M}_c$ with $\ell_s(j) > 0$ such that $\ell^+(j) + \ell_s(j) + p_t \le 4/3 + \varepsilon$, assign J_t to it. Otherwise check if $\exists\, M_j \in \mathcal{M}_c$ with $\ell_s(j) = 0$ such that $\ell^+(j) + p_t \le 4/3 + \varepsilon$. If so, assign J_t to such an M_j with the lowest $\ell^-(j)$; otherwise place J_t on an arbitrary $M_j \in \mathcal{M}_c$.

Fig. 3. The algorithm $\mathcal{A}_2(\varepsilon)$

that $\kappa = \lceil 2(2+1/\varepsilon')(2l-1)\rceil$ and $l = \lambda+2 = \lceil \log(\frac{3}{8} + \frac{1}{48\varepsilon'})\rceil + 2$. Hence $l = O(\log(1/\varepsilon))$ and $\kappa = O(1/\varepsilon \log(1/\varepsilon))$, which gives that $|C|$ is $1/\varepsilon^{O(\log(1/\varepsilon))}$.

Hence it suffices to show that, for any job sequence σ, $\mathcal{A}_2(\varepsilon)$ generates a schedule whose makespan is at most $(4/3 + \varepsilon)\mathrm{OPT}(\sigma)$, which we will do in the remainder of this section. More specifically we will prove that, for any σ, there exists a target configuration $c \in C$ that accurately models the large jobs arising in σ. We will refer to such a vector as a valid target configuration. Then we will show that the corresponding algorithm A_c builds a schedule with a makespan of at most $(4/3+\varepsilon)\mathrm{OPT}(\sigma)$.

We introduce some notation. Consider any job sequence σ. For any i, $1 \le i \le 2l-1$, let $n_i(\sigma)$ be the number of class-i jobs arising in σ, i. e. $n_i(\sigma)$ is the number of jobs J_t with $p_t \in I_i$. Furthermore, for any target configuration $c = (c_1, \dots, c_\mu) \in C$ and any i with $1 \le i \le 2l - 1$, let m_i be the number of class-i machines in c, i. e. $m_i = |\{c_j \in \{c_1, \dots, c_\mu\} : c_j = i\}|$. Let $\mu_1 = \sum_{i=1}^{l} m_i$ be the total number of class-i machines with $i \in \{1, \dots, l\}$. Similarly, $\mu_2 = \sum_{i=l+1}^{2l-1} m_i$ is the total number of class-i machines with $i \in \{l+1, \dots, 2l-1\}$. Given σ, vector $c \in C$ will be a valid target configuration if, for any $i = 1, \dots, 2l-1$, σ contains as many class-i jobs as specified in c and, moreover, if all the additional large jobs can be feasibly scheduled on the $m - \mu$ reserve machines. Recall that in a configuration c, any class-i machine with $1 \le i \le l$ is supposed to contain two class-i jobs. Formally, $c \in C$ is a *valid target configuration* if the following three conditions hold.

(i) For $i = 1, \dots, l$, there holds $2m_i \le n_i(\sigma)$.
(ii) For $i = l+1, \dots, 2l-1$, there holds $m_i \le n_i(\sigma)$.
(iii) $\lceil (\sum_{i=1}^{l} n_i(\sigma) - 2\mu_1)/2\rceil + \sum_{i=l+1}^{2l-1} n_i(\sigma) - \mu_2 \le m - \mu$

Conditions (i) and (ii) represent the constraint that σ contains as many class-i jobs as specified in c, $1 \le i \le 2l-1$. Condition (iii) models the requirement that extra large

jobs can be feasibly packed on the reserve machines. Here $\sum_{i=1}^{l} n_i(\sigma) - 2\mu_1$ is the extra number of class-i jobs with $i \in \{1, \dots, l\}$ in σ. Any two of these can be packed on one machine since the processing time of any of these jobs is upper bounded by $b_l \le 2/3 + 4\varepsilon'$. Hence two jobs incur a machine load of at most $4/3 + 8\varepsilon' = 4/3 + \varepsilon$. Analogously, $\sum_{i=l+1}^{2l-1} n_i(\sigma) - \mu_2$ is the extra number of class-i jobs with $i \in \{l+1, \dots, 2l-1\}$, which cannot be combined together because their processing times are greater than $2a_1 \ge 2/3 + 4\varepsilon'$.

In order to prove that, for any σ, there exists a valid target configuration we need two lemmas.

Lemma 3. *For any σ, there holds $\lceil \sum_{i=1}^{l} n_i(\sigma)/2 \rceil + \sum_{i=l+1}^{2l-1} n_i(\sigma) \le m$.*

Proof. Consider any optimal schedule S^* for σ and recall that we assume without loss of generality that $\text{OPT}(\sigma) = 1$. In S^* any machine containing a class-i job with $i \in \{l+1, \dots, 2l-1\}$ cannot contain an additional large job: The class-i job causes a load greater than $2a_1 \ge 2/3 + 4\varepsilon'$ and any additional large job, having a processing time greater than $1/3 + 2\varepsilon'$, would generate a total load greater than 1. Furthermore, any machine containing a class-i job with $i \in \{1, \dots, l\}$ can contain at most one additional job of the job classes $1, \dots, l$ because two further jobs would generate a total load greater than $3a_1 \ge 3(1/3 + 2\varepsilon') > 1$. ◻

Lemma 4. *For any $0 < \varepsilon' \le 1/8$, there holds $\kappa\lfloor (m-\mu)/(2l-1) \rfloor \ge m$ if $m \ge 2l/(\varepsilon')^2$.*

Proof. There holds

$$
\begin{aligned}
\kappa\lfloor (m-\mu)/(2l-1) \rfloor &\ge 2(2+\tfrac{1}{\varepsilon'})(2l-1) \cdot \lfloor (m-\mu)/(2l-1) \rfloor \\
&\ge 2(2+\tfrac{1}{\varepsilon'})(2l-1) \cdot ((m-\mu)/(2l-1) - 1) \\
&= 2(2+\tfrac{1}{\varepsilon'})(2l-1) \cdot \left(\frac{m - \lceil \frac{1+\varepsilon'}{1+2\varepsilon'} m \rceil}{2l-1} - 1 \right) \\
&\ge 2(2+\tfrac{1}{\varepsilon'})(2l-1) \cdot \left(\frac{\frac{\varepsilon'}{1+2\varepsilon'} m - 1}{2l-1} - 1 \right) \\
&= 2(2+\tfrac{1}{\varepsilon'})(2l-1) \cdot \left(\tfrac{\varepsilon' m - (1+2\varepsilon')2l}{(1+2\varepsilon')(2l-1)} \right) \\
&\ge m + m - (2/\varepsilon')(1+2\varepsilon')2l \\
&\ge m,
\end{aligned}
$$

where the last line follows because of $m \ge 2l/(\varepsilon')^2$ and $2l/(\varepsilon')^2 \ge (2/\varepsilon')(1+2\varepsilon')2l$, for any $\varepsilon' \le 1/8$. ◻

The next lemma establishes the existence of valid target configurations.

Lemma 5. *For any σ, there exists a valid target configuration $c \in C$ if $m \ge 2l/(\varepsilon')^2$.*

Proof. In this proof let $m_0 = \lfloor (m-\mu)/(2l-1) \rfloor$. Given σ, we first construct a vector $u \in U$. Lemma 3 implies that for any job class i, $1 \le i \le l$, there holds $\lceil n_i(\sigma)/2 \rceil \le m$. For any job class i, $l+1 \le i \le 2l-1$, there holds $n_i(\sigma) \le m$. By Lemma 4, $\kappa m_0 \ge m$,

which is equivalent to $m/m_0 \le \kappa$. For any i with $1 \le i \le l$, set $u_i = \lfloor n_i(\sigma)/(2m_0)\rfloor$. For any i with $l+1 \le i \le 2l-1$, set $u_i = \lfloor n_i(\sigma)/m_0\rfloor$. Then $u_i \in \{0,\dots,\kappa\}$, for $i = 1,\dots,2l-1$, and the resulting vector $u = (u_1,\dots .u_{2l-1})$ is element of U. We next show that the vector $c(u)$ constructed by $\mathcal{A}_2(\varepsilon)$ is a valid target configuration.

When $\mathcal{A}_2(\varepsilon)$ constructs $c(u)$, it first builds a vector $c'(u) = (c'_1,\dots,c'_{\mu'})$ of length $\mu' = \sum_{i=1}^{2l-1} u_i m_0$ containing exactly $u_i m_0$ entries with $c'_j = i$, for $i = 1,\dots,2l-1$. If $\mu' \ge \mu$, then $c(u)$ contains the first μ entries of $c'(u)$. If $\mu' < \mu$, then $c(u)$ is obtained from $c'(u)$ by adding $\mu - \mu'$ entries of value 0. In either case $c(u)$ contains at most $u_i m_0$ entries of values i, for $i = 1,\dots,2l-1$. Hence for the target configuration $c(u)$, there holds $m_i \le u_i m_0$, for $i = 1,\dots,2l-1$, where m_i is again the total number of class-i machines in $c(u)$.

If $i \in \{1,\dots,l\}$, then $m_i \le \lfloor n_i(\sigma)/(2m_0)\rfloor m_0 \le n_i(\sigma)/2$, which is equivalent to $2m_i \le n_i(\sigma)$. Similarly, if $i \in \{l+1,\dots,2l-1\}$, then $m_i \le \lfloor n_i(\sigma)/m_0\rfloor m_0 \le n_i(\sigma)$. Therefore, conditions (i) and (ii) defining valid target configurations are satisfied and we are left to verify condition (iii).

First assume $\mu' \ge \mu$. In this case the vector $c(u)$ contains no entries of value 0 and hence $\mu = \mu_1 + \mu_2$. Recall that $\mu_1 = \sum_{i=1}^{l} m_i$ is the total number of class-i machines with $i \in \{1,\dots,l\}$ specified in $c(u)$. Similarly, $\mu_2 = \sum_{i=l+1}^{2l-1} m_i$ is the total number of class-i machines with $i \in \{l+1,\dots,2l-1\}$. By Lemma 3, $\lceil \sum_{i=1}^{l} n_i(\sigma)/2\rceil + \sum_{i=l+1}^{2l-1} n_i(\sigma) \le m$. Subtracting the equation $\mu_1 + \mu_2 = \mu$, we obtain

$$\lceil \textstyle\sum_{i=1}^{l} n_i(\sigma)/2\rceil - \mu_1 + \sum_{i=l+1}^{2l-1} n_i(\sigma) - \mu_2 \le m - \mu.$$

There holds $\lceil \sum_{i=1}^{l} n_i(\sigma)/2\rceil - \mu_1 = \lceil (\sum_{i=1}^{l} n_i(\sigma) - 2\mu_1)/2\rceil$ because μ_1 is an integer. Hence condition (iii) defining valid target configurations is satisfied.

It remains to study the case $\mu' < \mu$. For any i with $i \in \{l+1,\dots,2l-1\}$, there holds $u_i = \lfloor n_i(\sigma)/m_0\rfloor$ and hence $u_i > n_i(\sigma)/m_0 - 1$, which is equivalent to $n_i(\sigma) < (u_i+1)m_0$. Hence

$$\textstyle\sum_{i=l+1}^{2l-1} n_i(\sigma) < \sum_{i=l+1}^{2l-1} (u_i+1)m_0 = \sum_{i=l+1}^{2l-1} u_i m_0 + (l-1)m_0.$$

The sum $\sum_{i=l+1}^{2l-1} u_i m_0 = \sum_{i=l+1}^{2l-1} u_i \lfloor (m-\mu)/(2l-1)\rfloor$ is the total number of entries c'_j with $c'_j \in \{l+1,\dots,2l-1\}$ in $c'(u)$. Since $\mu' < \mu$, none of these entries is deleted when $c(u)$ is derived from $c'(u)$. Hence $\sum_{i=l+1}^{2l-1} u_i m_0 = \mu_2$ is the total number of class-i machines with $i \in \{l+1,\dots,2l-1\}$ specified in $c(u)$. We conclude

$$\textstyle\sum_{i=l+1}^{2l-1} n_i(\sigma) \le \mu_2 + (l-1)m_0. \tag{6}$$

For any i with $i \in \{1,\dots,l\}$, there holds $u_i = \lfloor n_i(\sigma)/(2m_0)\rfloor$ and hence $u_i > n_i(\sigma)/(2m_0) - 1$. This implies $n_i(\sigma)/2 < (u_i+1)m_0$. Since $(u_i+1)m_0$ is an integer we obtain $n_i(\sigma)/2 \le (u_i+1)m_0 - 1$. Thus

$$\lceil \textstyle\sum_{i=1}^{l} n_i(\sigma)/2\rceil \le \sum_{i=1}^{l} n_i(\sigma)/2 + 1 \le \sum_{i=1}^{l} (u_i+1)m_0 = \mu_1 + l m_0. \tag{7}$$

Again $\sum_{i=1}^{l} u_i m_0 = \mu_1$ because $c'(u)$ contains exactly $\sum_{i=1}^{l} u_i m_0$ entries c'_j with $c'_j \in \{1,\dots,l\}$ and all of these entries are contained in $c(u)$ representing class-i machines for

$i \in \{1, \dots, l\}$. Inequalities (6) and (7) together with the identity $m_0 = \lfloor (m-\mu)/(2l-1) \rfloor$ imply

$$\lceil \sum_{i=1}^{l} n_i(\sigma)/2 \rceil - \mu_1 + \sum_{i=l+1}^{2l-1} n_i(\sigma) - \mu_2 \leq (2l-1)\lfloor (m-\mu)/(2l-1) \rfloor \leq m - \mu.$$

Since again $\lceil \sum_{i=1}^{l} n_i(\sigma)/2 \rceil - \mu_1 = \lceil (\sum_{i=1}^{l} n_i(\sigma) - 2\mu_1)/2 \rceil$, condition (iii) defining valid target configurations holds. □

We next analyze the scheduling steps of $\mathcal{A}_2(\varepsilon)$.

Lemma 6. *Let A_c be any algorithm of $\mathcal{A}_2(\varepsilon)$ processing a job sequence σ. At any time there exists at most one machine $M_j \in \mathcal{M}_c$ with $\ell_s(j) > 0$ and $\ell^-(j) + \ell_s(j) < 1 + \varepsilon'$ in the schedule maintained by A_c.*

Proof. Consider any point in time while A_c sequences σ. Suppose that there exists a machine $M_j \in \mathcal{M}_c$ with $\ell_s(j) > 0$ and $\ell^-(j) + \ell_s(j) < 1 + \varepsilon'$. We show that if a small job J_t arrives and A_c assigns it to a machine $M_{j'} \in \mathcal{M}_c$ with $\ell_s(j') = 0$, then $\ell^-(j') + p_t > 1 + \varepsilon'$ so that no new machine with the property specified in the lemma is generated. A first observation is that M_j is not a class-l machine because in this case $\ell^-(j)$ would be $2a_l = 2b_{l-1} = 1 + 2\varepsilon'$. Also, if $M_{j'}$ is a class-l machine, there is nothing to show because, again, in this case $\ell^-(j') \geq 1 + 2\varepsilon'$.

So assume that A_c assigns J_t to a machine $M_{j'} \in \mathcal{M}_c$, which is not a class-l machine, and $\ell_s(j') = 0$ prior to the assignment. We first show that $\ell^-(j') \geq \ell^-(j)$. Consider the scheduling step in which A_c assigned the first small job $J_{t'}$ to M_j. Since M_j is not a class-l machine $\ell^+(j) = 2b_i$, for some $i \in \{1, \dots, l-1\}$ and the assignment of $J_{t'}$ to M_j led to a load of at most $\ell^+(j) + p_{t'} \leq 1 + 2\varepsilon' + 1/3 + 2\varepsilon' = 4/3 + 4\varepsilon' < 4/3 + \varepsilon$. Since $M_{j'}$ is not a class-l machine either, $J_{t'}$ could have also been assigned to $M_{j'}$ incurring a resulting load of at most $\ell^+(j') + p_{t'} < 4/3 + \varepsilon$ on this machine. Note that when an algorithm A_c cannot assign a small job to a machine $M_j \in \mathcal{M}_c$ with $\ell_s(j) > 0$ and instead has to resort to machines $M_k \in \mathcal{M}_c$ with $\ell_s(k) = 0$, it chooses a machine having the smallest $\ell^-(k)$ value. We conclude $\ell^-(j) \leq \ell^-(j')$.

Next consider the assignment of J_t. Algorithm A_c would prefer to place J_t on M_j as it already contains small jobs. Since this is impossible, there holds $\ell^+(j) + \ell_s(j) + p_t > 4/3 + \varepsilon$ and thus $p_t > 4/3 + 8\varepsilon' - \ell^+(j) - \ell_s(j)$. Since by assumption $\ell^-(j) + \ell_s(j) < 1 + \varepsilon'$ it follows $p_t > 1/3 + 7\varepsilon' - \ell^+(j) + \ell^-(j)$. Suppose that $\ell^+(j) = 2b_i$, for some $i \in \{1, \dots, l-1\}$. Then $\ell^-(j) = 2a_i$. Since $\ell^-(j') \geq \ell^-(j)$ we obtain

$$\begin{aligned}
\ell^-(j') + p_t &\geq 1/3 + 7\varepsilon' + \ell^-(j) - \ell^+(j) + \ell^-(j) \\
&\geq 1/3 + 7\varepsilon' + 2\left(\tfrac{1}{12} + \tfrac{3}{2}\varepsilon'\right)\left(\tfrac{1}{2^{\lambda+1-i}} - \tfrac{1}{2^{\lambda-i}}\right) \\
&\quad +2/3 - 4\varepsilon' + 2\left(\tfrac{1}{12} + \tfrac{3}{2}\varepsilon'\right)\tfrac{1}{2^{\lambda+1-i}} \\
&= 1 + 3\varepsilon' > 1 + \varepsilon',
\end{aligned}$$

as desired. □

The following lemmas focus on algorithms A_c such that c is a valid target configuration for σ.

Lemma 7. *Let σ be any job sequence and A_c be an algorithm such that c is a valid target configuration for σ. Let $m \geq 2l/(\varepsilon')^2$. Consider any point in time during the scheduling process. If the schedule of A_c contains at most one machine $M_j \in \mathcal{M}_c$ with $\ell^-(j) + \ell_s(j) < 1 + \varepsilon'$, then no further small job can arrive.*

Proof. Since c is a valid target configuration for σ, the job sequence contains as many class-i jobs, for any $i \in \{1, \ldots, l\}$, as indicated by c. Hence the total processing time of large jobs in σ is lower bounded by $\sum_{j=1}^{\mu} \ell^-(j)$. Hence the total processing time of jobs in σ is at least $\sum_{j=1}^{\mu}(\ell^-(j) + \ell_s(j))$, where the machine loads due to small jobs may be considered at an arbitrary point in time. Hence if there exists a time such that $\ell_s(j) + \ell^-(j) < 1 + \varepsilon'$ for at most one $M_j \in \mathcal{M}_c$, we obtain

$$\begin{aligned}\textstyle\sum_{j=1}^{\mu}(\ell^-(j) + \ell_s(j)) &\geq (1 + \varepsilon')(\mu - 1) \geq (1 + \varepsilon')(\tfrac{1+\varepsilon'}{1+2\varepsilon'}m - 1) \\ &= m + \tfrac{(\varepsilon')^2}{1+2\varepsilon'}m - (1 + \varepsilon') \geq m.\end{aligned}$$

The last inequality holds because $m \geq 2l/(\varepsilon')^2 \geq 2/(\varepsilon')^2 \geq (1 + \varepsilon')(2\varepsilon' + 1)/(\varepsilon')^2$, for any $\varepsilon' \leq 1/8$. Hence no further small job can arrive. □

Lemma 8. *Let σ be any job sequence and A_c be an algorithm such that c is a valid target configuration for σ. Let $m \geq 2l/(\varepsilon')^2$. Then in the final schedule constructed by A_c, each machine in $\mathcal{M}_c$ has a load of at most $4/3 + \varepsilon$.*

Proof. We consider the scheduling steps in which A_c assigns a job J_t to a machine in $\mathcal{M}_c$. First suppose that J_t is large. Let J_t be a class-i job, where $1 \leq i \leq 2l - 1$. If J_t is assigned to an $M_j \in \mathcal{M}_c$, then M_j must be an admissible class-i machine, i. e. prior to the assignment of J_t it contains fewer class-i jobs as specified by the target configuration. This implies that for any machine $M_j \in \mathcal{M}_c$, its load due to large jobs is always at most $\ell^+(j)$. The latter value is upper bounded by $2b_l \leq 2(2/3 + 4\varepsilon') = 4/3 + 8\varepsilon' = 4/3 + \varepsilon$. Hence, in order to establish the lemma it suffices to show that whenever a small job is assigned to a machine $M_j \in \mathcal{M}_c$, the resulting load $\ell^+(j)+\ell_s(j)$ on M_j is at most $4/3 + \varepsilon$.

Suppose on the contrary that a small job J_t arrives and A_c schedules it on a machine in $\mathcal{M}_c$ such that the resulting load is greater than $4/3 + \varepsilon$. Algorithm A_c first tries to place J_t on a machine $M_j \in \mathcal{M}_c$ with $\ell_s(j) > 0$, which has already received small jobs. By Lemma 6, among these machines there exists at most one having the property that $\ell^-(j) + \ell_s(j) < 1 + \varepsilon'$. Since an assignment to those machines is impossible without exceeding a load of $4/3 + \varepsilon$, A_c tries to place J_t on a machine $M_j \in \mathcal{M}_c$ with $\ell_s(j) = 0$. Since this is also impossible without exceeding a load of $4/3 + \varepsilon$, any $M_j \in \mathcal{M}_c$ with $\ell_s(j) = 0$ must be a class-l machine. This holds true because for any class-i machine with $i \neq l$, there holds $\ell^+(j) \leq 2b_{l-1} \leq 1 + 2\varepsilon'$ and an assignment of a small job would result in a total load of at most $1 + 2\varepsilon' + 1/3 + 2\varepsilon' < 4/3 + \varepsilon$. Observe that any class-l machine has a targeted minimal load of $2a_l = 2b_{l-1} \geq 1 + 2\varepsilon' > 1 + \varepsilon'$.

We conclude that immediately before the assignment of J_t the schedule of A_c contains at most one machine $M_j \in \mathcal{M}_c$ with $\ell^-(j) + \ell_s(j) < 1 + \varepsilon'$. Lemma 7 implies that the incoming job J_t cannot be small, and we obtain a contradiction. □

Lemma 9. *Let σ be any job sequence and A_c be an algorithm such that c is a valid target configuration for σ. Then in the final schedule constructed by A_c, each machine in $\mathcal{M}_r$ has a load of at most $4/3 + \varepsilon$.*

Proof. Algorithm A_c assigns only large jobs to machines in $\mathcal{M}_r$. A first observation is that whenever there exists an $M_j \in \mathcal{M}_r$ that contains only one class-i job with $i \in \{1, \ldots, l\}$ but no further jobs, then an incoming class-i' job with $i' \in \{1, \ldots, l\}$ will not be assigned to an empty machine. This holds true because the two jobs can be combined, which results in a total load of at most $2b_l \le 4/3 + 8\varepsilon' = 4/3 + \varepsilon$.

The observation implies that at any time while A_c processes σ, the number of machines of $\mathcal{M}_r$ containing at least one job is upper bounded by $\lceil n_1/2 \rceil + n_2$. Here n_1 denotes the total number of class-i jobs with $i \in \{1, \ldots, l\}$ that have been assigned to machines of $\mathcal{M}_r$ so far. Analogously, n_2 is the total number of class-i jobs with $i \in \{l+1, \ldots, 2l-1\}$ currently residing on machines in $\mathcal{M}_r$. Since c is a valid target configuration for σ conditions (i) and (ii) defining those configurations imply $0 \le \sum_{i=1}^{l} n_i(\sigma) - 2\mu_1$ and $0 \le \sum_{i=l+1}^{2l-1} n_i(\sigma) - \mu_2$. Moreover, since A_c assigns large jobs preferably to machines in $\mathcal{M}_c$, there holds $n_1 \le \sum_{i=1}^{l} n_i(\sigma) - 2\mu_1$ and $n_2 \le \sum_{i=l+1}^{2l-1} n_i(\sigma) - \mu_2$. By condition (iii) defining valid target configurations, $\lceil (\sum_{i=1}^{l} n_i(\sigma) - 2\mu_1)/2 \rceil + \sum_{i=l+1}^{2l-1} n_i(\sigma) - \mu_2 \le m - \mu$. Hence, while $n_2 < \sum_{i=l+1}^{2l-1} n_i(\sigma) - \mu_2$ there holds $\lceil n_1/2 \rceil + n_2 < m - \mu$ and thus exists an empty machine $\mathcal{M}_r$ to which an incoming class-i jobs with $i \in \{l+1, \ldots, 2l-1\}$ can be assigned. Similarly, while $n_1 < \sum_{i=1}^{l} n_i(\sigma) - 2\mu_1$, there must exist an empty machine or a machine containing only one class-i' job with $i' \in \{1, \ldots, l\}$ to which in incoming class-i job with $i \in \{1, \ldots, l\}$ can be assigned. In either case, the assignment generates a load of at most $4/3 + \varepsilon$ on the selected machine. $\square$

Theorem 3 now follows from Lemmas 5, 8 and 9.

5 Algorithms for MPS

We derive our algorithms for MPS. The strategies are obtained by simply combining $\mathcal{A}^*(\varepsilon)$, presented in Section 2, with $\mathcal{A}_1(\varepsilon)$ and $\mathcal{A}_3(\varepsilon)$. In order to achieve a precision of ε in the competitive ratio, the strategies are combined with a precision of $\varepsilon/2$ in its parameters. For any $0 < \varepsilon \le 1$, let $\mathcal{A}_3^*(\varepsilon)$ be the algorithm obtained by executing $\mathcal{A}_3(\varepsilon/2)$ in $\mathcal{A}^*(\varepsilon/2)$. For any $0 < \varepsilon \le 1$, let $\mathcal{A}_1^*(\varepsilon)$ be the algorithm obtained by executing $\mathcal{A}_1(\varepsilon/2)$ in $\mathcal{A}^*(\varepsilon/2)$.

Corollary 2. *$\mathcal{A}_3^*(\varepsilon)$ is a $(4/3 + \varepsilon)$-competitive algorithm for MPS and uses no more than $1/\varepsilon^{O(\log(1/\varepsilon))}$ schedules, for any $0 < \varepsilon \le 1$.*

Proof. Theorem 1 and Corollary 1 imply that $\mathcal{A}_3^*(\varepsilon)$ is $(4/3 + \varepsilon)$-competitive, for any $0 < \varepsilon \le 1$, and that the total number of schedules is the product of $1/\varepsilon^{O(\log(1/\varepsilon))}$ and $\lceil \log(1+12\rho/\varepsilon)/\log(1+\varepsilon/(6\rho)) \rceil$, where $\rho = 4/3+\varepsilon/2$. By the Taylor series for $\ln(1+x)$, $-1 < x \le 1$, we obtain $\ln(1 + x) \ge x/2$, for any $0 < x \le 1$. Hence the second term of the product is $1/\varepsilon^{O(1)}$. $\square$

Corollary 3. *$\mathcal{A}_1^*(\varepsilon)$ is a $(1+\varepsilon)$-competitive algorithm for MPS and uses no more than $(m/\varepsilon)^{O(\log(1/\varepsilon)/\varepsilon)}$ schedules, for any $0 < \varepsilon \le 1$.*

Proof. By Theorems 1 and 2 algorithm $\mathcal{A}_1^*(\varepsilon)$ is $(1+\varepsilon)$-competitive, for any $0 < \varepsilon \le 1$. The total number of schedules is the product of $(\lfloor 4m/\varepsilon \rfloor + 1)^{\lceil \log(4/\varepsilon)/\log(1+\varepsilon/4) \rceil}$ and $\lceil \log(1 + 12\rho/\varepsilon)/\log(1 + \varepsilon/(6\rho)) \rceil$, where $\rho = 1 + \varepsilon/2$. Again, by the Taylor series, $\ln(1+x) \ge x/2$, for any $0 < x \le 1$. Hence both terms of the product are upper bounded by $(m/\varepsilon)^{O(\log(1/\varepsilon)/\varepsilon)}$. □

6 Lower bounds

We develop lower bounds that apply to both MPS and MPS_{opt}.

Theorem 4. *Let $\mathcal{A}$ be a deterministic online algorithm for MPS or MPS_{opt}. If $\mathcal{A}$ attains a competitive ratio smaller than $4/3$, then it must maintain at least $\lfloor m/3 \rfloor + 1$ schedules.*

Proof. Let $\mathcal{A}$ be any deterministic online algorithm for MPS or MPS_{opt} that maintains at most $\lfloor m/3 \rfloor$ schedules. We show that $\mathcal{A}$'s competitive ratio is at least $4/3$. To this end we construct an adversarial job sequence σ such that each schedule maintained by $\mathcal{A}$ has a makespan of at least $4/3 \cdot \text{OPT}(\sigma)$.

The job sequence σ is composed of two subsequences σ_1 and σ_2, i. e. $\sigma = \sigma_1\sigma_2$. Subsequence σ_1 consists of m jobs of processing time $1/3$ each. Subsequence σ_2 will consist of jobs having a processing time of either 2/3 or 1. The exact number of these jobs depends on the schedules constructed by $\mathcal{A}$ and will be determined later.

Consider the schedules that $\mathcal{A}$ may have built after all jobs of σ_1 have been assigned. Each such schedule contains m jobs of processing time 1/3. For the moment we concentrate on schedules in which each machine contains either zero, one or three jobs, i. e. there exists no machine containing two or more than three jobs. Each such schedule S can be represented by a pair (m_1, m_3), where m_1 denotes the number of machines containing exactly one job and m_3 is the number of machines containing three jobs. Here m_1 and m_3 are non-negative integers such that $m_1 + 3m_3 = m$. Let $P = \{(m_1, m_3) \mid m_1, m_3 \in \mathbb{N}_0 \text{ and } m_1 + 3m_3 = m\}$ be the set of all these pairs. Set P has $\lfloor m/3 \rfloor + 1$ elements because m_3 can take any value between 0 and $\lfloor m/3 \rfloor$ and $m_1 = m - 3m_3$. Let S be an arbitrary schedule containing m jobs of processing time 1/3 and $(m_1, m_3) \in P$. We say that S is an (m_1, m_3)-*schedule* if the number of machines containing exactly one job equals m_1 and the number of machines containing exactly three jobs equals m_3.

Let $\mathcal{S}$ be the set of schedules constructed by $\mathcal{A}$ when the entire subsequence σ_1 has arrived. By assumption $\mathcal{A}$ maintains at most $\lfloor m/3 \rfloor$ schedules, i. e. $|\mathcal{S}| \le \lfloor m/3 \rfloor$. Hence there must exist a pair $(m_1^*, m_3^*) \in P$ such that no schedule of $\mathcal{S}$ is an (m_1^*, m_3^*)-schedule. On the other hand, let $\mathcal{S}^*$ be an (m_1^*, m_3^*)-schedule. In $\mathcal{S}^*$ we number the machines in order of non-decreasing load such that $\ell^*(1) \le \ldots \le \ell^*(m)$. Schedule $\mathcal{S}^*$ contains $m - m_3^*$ machines with a load smaller than 1 and, in particular, $m - m_1^* - m_3^*$ empty machines.

Now the subsequence σ_2 consists of $m - m_3^*$ jobs, where the j-th job has a processing time of $1 - \ell^*(j)$, for $j = 1, \ldots, m - m_3^*$. Hence σ_2 contains $m - m_1^* - m_3^*$ jobs of

processing time 1 followed by m_1^* jobs of processing time $2/3$. Obviously, the makespan of an optimal schedule for σ is 1: The jobs of σ_1 are sequenced so that an (m_1^*, m_3^*)-schedule is obtained. Again, after σ_1 has arrived, the machines are numbered in order of non-decreasing load. While σ_2 arrives, the j-th job is assigned to machine M_j, having a load of $\ell^*(j)$, for $j = 1, \ldots, m - m_3^*$.

In the remainder of this proof we consider any schedule $S \in \mathcal{S}$ and show that after σ_2 has been sequenced, the resulting makespan is at least 4/3. This establishes the theorem. So let $S \in \mathcal{S}$ be any schedule and recall that S contains m jobs of processing time 1/3 each. If in S there exists a machine that contains at least four of these jobs, then the makespan is already 4/3 and there is nothing to show. Therefore, we restrict ourselves to the case that every machine in S contains at most three jobs. Again we number the machines in S in order of non-decreasing load so that $\ell(1) \leq \ldots \leq \ell(m)$. Consider the (m_1^*, m_3^*)-schedule $\mathcal{S}^*$ in which the machines loads satisfy $\ell^*(1) \leq \ldots \leq \ell^*(m)$. There must exist a machine M_{j_0}, $1 \leq j_0 \leq m$, such that $\ell(j_0) > \ell^*(j_0)$: For, if $\ell(j_0) \leq \ell^*(j_0)$ held for all $j = 1, \ldots, m$, then $\ell(j_0) = \ell^*(j_0)$ for all $j = 1, \ldots, m$ because S and S^* both contain jobs with a total processing time of $m/3$. Thus S would be an (m_1^*, m_3^*)-schedule and we obtain a contradiction. The last m_3^* machines in S^* have a load of 1. It follows that $j_0 \leq m - m_3^*$ because otherwise M_{j_0} in S contained at least four jobs. The property $\ell(j_0) > \ell^*(j_0)$ implies $\ell(j_0) \geq \ell^*(j_0) + 1/3$ because S and S^* only contain jobs of processing time $1/3$.

We finally show that sequencing of σ_2 leads to a makespan of at least $4/3$ in S. If $\mathcal{A}$ assigns two jobs of σ_2 to the same machine, then the resulting machine load is at least 4/3 because each job of σ_2 has a processing time of at least $2/3$. So assume that $\mathcal{A}$ assigns the jobs of σ_2 to different machines. The first j_0 jobs of σ_2 each have a processing time of at least $1 - \ell^*(j_0)$ because the jobs arrive in order of non-increasing processing times. In S there exist at most $j_0 - 1$ machines having a load strictly smaller than $\ell(j_0)$. Hence, after the first j_0 jobs have been scheduled in S, there exists a machine having a load of at least $\ell(j_0) + 1 - \ell^*(j_0) \geq \ell^*(j_0) + 1/3 + 1 - \ell^*(j_0) = 4/3$. This concludes the proof. □

The next theorem gives a lower bound on the number of schedules required by a $(1+\varepsilon)$-competitive algorithm, where $0 < \varepsilon < 1/4$. It implies that, for any fixed ε, the number asymptotically depends on $m^{\Omega(1/\varepsilon)}$, as m increases. For instance, any algorithm with a competitive ratio smaller than $1 + 1/12$ requires $\Omega(m^2)$ schedules. Any algorithm with a competitive ratio smaller than $1 + 1/16$ needs $\Omega(m^3)$ schedules.

Theorem 5. *Let $\mathcal{A}$ be a deterministic online algorithm for MPS or MPS$_{\text{opt}}$. If $\mathcal{A}$ attains a competitive ratio smaller than $1 + \varepsilon$, where $0 < \varepsilon \leq 1/4$, then it must maintain at least $\binom{m'+h-1}{h-1}$ schedules, where $m' = \lfloor m/2 \rfloor$ and $h = \lfloor 1/(4\varepsilon) \rfloor$. The binomial coefficient increases as ε decreases and is at least $\Omega((\varepsilon m)^{\lfloor 1/(4\varepsilon) \rfloor - 1/2}/\sqrt{m})$.*

Proof. We extend the proof of Theorem 4. Let $0 < \varepsilon \leq 1/4$. Furthermore, let m' and h be defined as in the theorem. There holds $h \geq 1$. Let $\varepsilon' = 1/(4h)$ and note that $\varepsilon' \geq \varepsilon$. We will define a set M whose cardinality is at least $\binom{m'+h-1}{h-1}$, and show that if $\mathcal{A}$ maintains less than $|M|$ schedules, then its competitive ratio is at least $1 + \varepsilon'$.

We specify a job sequence σ and first assume that m is even. Later we will describe how to adapt σ if m is odd. Again σ is composed of two partial sequences σ_1 and

σ_2 so that $\sigma = \sigma_1\sigma_2$. Subsequence σ_1 consists of mh jobs of processing time ε' each. Subsequence σ_2 depends on the schedules constructed by $\mathcal{A}$ and will be specified below. Consider the possible schedules after σ_1 has been sequenced on the m machines. We restrict ourselves to schedules having the following property: Each machine has a load of exactly 1 or a load that is at most $1/2 - \varepsilon'$. Observe that each machine of load 1 contains $1/\varepsilon'$ jobs. Each machine of load at most $1/2 - \varepsilon'$ contains up to $2h - 1$ jobs because $(2h-1)\varepsilon' = 2h/(4h) - \varepsilon' = 1/2 - \varepsilon'$. Therefore any schedule with the stated property can be described by a vector $\vec{m} = (m_0, \ldots, m_{2h})$, where m_{2h} is the number of machines having a load of 1 and m_i is the number of machines containing exactly i jobs, for $i = 0, \ldots, 2h-1$. The vector $\vec{m}$ satisfies $\sum_{i=0}^{2h} m_i = m$ and $(1/\varepsilon')m_{2h} + \sum_{i=1}^{2h-1} im_i = mh$. The last equation specifies the constraint that the schedule contains mh jobs. Let M be the set of all these vectors, i. e.

$$M = \{(m_0, \ldots, m_{2h}) \in \mathbb{N}_0^{2h+1} \mid \textstyle\sum_{i=0}^{2h} m_i = m \text{ and } (1/\varepsilon')m_{2h} + \sum_{i=1}^{2h-1} im_i = mh\}.$$

We remark that each $\vec{m} \in M$ uniquely identifies one schedule with our desired property. Let S be any schedule containing exactly mh jobs of processing time ε' and $\vec{m} = (m_0, \ldots, m_{2h}) \in M$. We say that S is an *$\vec{m}$-schedule* if in S there exist m_{2h} machines of load 1 and m_i machines containing exactly i jobs, for $i = 0, \ldots, 2h-1$.

Now suppose that $\mathcal{A}$ maintains less than $|M|$ schedules. Let $\mathcal{S}$ be the set of schedules constructed by $\mathcal{A}$ after all jobs of σ_1 have arrived. Since $|\mathcal{S}| < |M|$ there must exist an $\vec{m}^* = (m_0^*, \ldots, m_{2h}^*) \in M$ such that no schedule of $\mathcal{S}$ is an $\vec{m}^*$-schedule. Let S^* be an $\vec{m}^*$-schedule in which machines are numbered in order of non-decreasing load such that $\ell^*(1) \le \ldots \le \ell^*(m)$. Subsequence σ_2 consists of $m - m_{2h}^*$ jobs, where job j has a processing time of $1 - \ell^*(j)$, for $j = 1, \ldots, m - m_{2h}^*$. Hence σ_2 consists of m_i^* jobs of processing time $1 - i\varepsilon'$, for $i = 0, \ldots, 2h-1$. These jobs arrive in order of non-increasing processing time. Each job has a processing time of at least $1/2 + \varepsilon'$ because $1 - (2h-1)\varepsilon' = 1 - (2h/4h - \varepsilon') = 1/2 + \varepsilon'$. The makespan of an optimal schedule for σ is 1. The jobs of σ_1 are sequenced so that an $\vec{m}^*$-schedule is obtained. Machines are again numbered in order of non-decreasing load. Then, while the jobs of σ_2 arrive, the j-th job of the subsequence is assigned to machine M_j in S^*, $1 \le j \le m - m_{2h}^*$.

We next show that after $\mathcal{A}$ has sequenced σ_2, each of its schedules has a makepan of at least $1+\varepsilon'$. So consider any $S \in \mathcal{S}$ and, as always, number the machines in order of non-decreasing load such that $\ell(1) \le \ldots \le \ell(m)$. If in S there exists a machine that has a load of at least $1 + \varepsilon'$ and hence contains at least $1/\varepsilon' + 1$ jobs, then there is nothing to show. So assume that each machine in S contains at most $1/\varepsilon'$ jobs and thus has a load of at most 1. We study the assignment of the jobs of σ_2 to S. If $\mathcal{A}$ places two jobs of σ_2 on the same machine, then we are done because each job has a processing time of at least $1/2 + \varepsilon'$. Therefore we focus on the case that $\mathcal{A}$ assigns the jobs of σ_2 to different machines.

Schedules S and S^* both contain jobs of total processing time $mh\varepsilon'$. Since S is not an $\vec{m}^*$-schedule there must exist a j_0, $1 \le j_0 \le m$, such that $\ell(j_0) > \ell^*(j_0)$ and hence $\ell(j_0) \ge \ell^*(j_0) + \varepsilon'$. Each machine in S has a load of at most 1 while the last $m - m_{2h}^*$ machines in S^* have a load of exactly 1. This implies $j_0 \le m - m_{2h}^*$. The first j_0 jobs of σ_2 each have a processing time of at least $1 - \ell^*(j_0)$. However, there exist at

most $j_0 - 1$ machines in S having a load strictly smaller than $\ell^*(j_0)$. Hence after $\mathcal{A}$ has sequenced the first j_0 jobs of σ_2 there must exist a machine in S with a load of at least $\ell(j_0) + 1 - \ell^*(j_0) \geq \ell^*(j_0) + \varepsilon' + 1 - \ell^*(j_0) = 1 + \varepsilon'$.

So far we have assumed that m is even. If m is odd, we can easily modify σ. The first job of σ is a job of processing time 1. Then σ_1 and σ_2 follow. These subsequences are defined as above, where m is replaced by the even number $m - 1$. In this case

$$M = \{(m_0, \dots, m_{2h}) \in \mathbb{N}_0^{2h-1} \mid \textstyle\sum_{i=0}^{2h} m_i = m - 1 \text{ and } \\ (1/\varepsilon')m_{2h} + \textstyle\sum_{i=1}^{2h-1} i m_i = (m-1)h\}.$$

The analysis presented above carries over because the first job of σ, having a processing time of 1, must be scheduled on a separate machine and cannot be combined with any job of σ_1 or σ_2 if a competitive ratio smaller than $1 + \varepsilon'$ is to be attained.

We next lower bound the cardinality of M. Again we first focus on the case that m is even. In the definition of M the critical constraint is $(1/\varepsilon')m_{2h} + \sum_{i=1}^{2h-1} i m_i = mh$, which implies that not every vector of $\{0, \dots, m\}^{2h+1}$ represents a schedule that can be built of mh jobs. In particular, the vector $(0, \dots, 0, m)$ of length $2h + 1$ would require $m/\varepsilon' = 4h$ jobs. Therefore, we introduce a set M' and show $|M'| \leq |M|$. Set M' contains vectors of length $2h + 1$ in which the first $h + 1$ entries as well as the last one are equal to 0. The other entries sum to at most $m/2$, i. e.

$$M' = \{(0, \dots, 0, m'_{h+1}, \dots, m'_{2h-1}, 0) \in \mathbb{N}_0^{2h+1} \mid \textstyle\sum_{i=1}^{h-1} m'_{h+i} \leq m/2\}.$$

We show that each $\vec{m}' \in M'$ can be mapped to a $\vec{m} \in M$. The mapping has the property that any two different vectors of M' are mapped to different vectors of M. This implies $|M'| \leq |M|$.

Consider any $\vec{m}' = (0, \dots, 0, m'_{h+1}, \dots, m'_{2h-1}, 0) \in M'$. Let $\vec{m} = (m_0, \dots, m_{2h})$ be defined as follows. For $i = h + 1, \dots, 2h$, let $m_i = m'_i$. For $i = 0, \dots, h - 1$, let $m_i = m_{2h-i}$. Finally, let $m_h = m - 2\sum_{i=1}^{h-1} m_i$. Note that $m_0 = m_{2h} = 0$. We next show that $\vec{m} \in M$. There holds $\sum_{i=0}^{2h} m_i = \sum_{i=1}^{2h-1} m_i = 2\sum_{i=1}^{h-1} m_i + m_h = m$. Furthermore,

$$\begin{aligned} m_{2h}/\varepsilon' + \sum_{i=0}^{2h-1} i m_i &= \sum_{i=1}^{2h-1} i m_i = \sum_{i=1}^{h-1} (i + 2h - i) m_i + h m_h \\ &= 2h \sum_{i=1}^{h-1} m_i + h(m - 2\sum_{i=1}^{h-1} m_i) = mh. \end{aligned}$$

It follows, as desired, $\vec{m} \in M$. Note that the last h entries of $\vec{m}$ are identical to the last h entries of $\vec{m}'$. Hence no two vectors of M' that differ in at least one entry are mapped to the same vector of M. Hence $|M'| \leq |M|$. If the number m of machines is odd, then in the definition of M' the entries of a vector sum to at most $(m-1)/2$. The rest of the construction and analysis is the same. Thus, for a general number m of machines

$$M' = \{(0, \dots, 0, m'_{h+1}, \dots, m'_{2h-1}, 0) \mid m'_i \in \mathbb{N}_0 \text{ and } \textstyle\sum_{i=1}^{h-1} m'_{h+i} \leq \lfloor m/2 \rfloor\}.$$

This set contains exactly $\binom{m'+h-1}{h-1}$ elements, where again $m' = \lfloor m/2 \rfloor$. In the remainder of this proof we lower bound this binomial coefficient.

There holds $\sqrt{2\pi e}(k/e)^{k+1/2} \le k! \le 2\sqrt{2\pi e}(k/e)^{k+1/2}$ for any $k \in \mathbb{N}$ by Stirling's approximation [17]. Hence

$$\begin{aligned}\binom{m'+h-1}{h-1} = \frac{(m'+h-1)!}{m'!(h-1)!} &\ge \frac{(m'+h-1)^{m'+h-1/2}}{4\sqrt{2\pi}(m')^{m'+1/2}(h-1)^{h-1/2}} \\ &= \frac{1}{4\sqrt{2\pi m'}}\left(1+\frac{h-1}{m'}\right)^{m'}\left(1+\frac{m'}{h-1}\right)^{h-1/2} \\ &> \frac{1}{4\sqrt{2\pi m'}}\left(1+\frac{m/2-1/2}{1/(4\varepsilon)}\right)^{h-1/2}.\end{aligned}$$

The last expression is $\Omega((\varepsilon m)^{\lfloor 1/(4\varepsilon)\rfloor-1/2}/\sqrt{m})$. □

References

1. S. Albers. Better bounds for online scheduling. *SIAM J. Comput.*, 29:459-473, 1999.
2. S. Albers and M. Hellwig. On the value of job migration in online makespan minimization. *Proc. 20th Annual European Symposium on Algorithms*, Springer LNCS 7501, 84–95, 2012.
3. E. Angelelli, A.B. Nagy, M.G. Speranza and Z. Tuza. The on-line multiprocessor scheduling problem with known sum of the tasks. *Journal of Scheduling*, 7:421–428, 2004.
4. E. Angelelli, M.G. Speranza and Z. Tuza. Semi-on-line scheduling on two parallel processors with an upper bound on the items. *Algorithmica*, 37:243–262, 2003.
5. E. Angelelli, M.G. Speranza and Z. Tuza. New bounds and algorithms for on-line scheduling: two identical processors, known sum and upper bound on the tasks. *Discrete Mathematics & Theoretical Computer Science*, 8:1–16, 2006.
6. Y. Azar. On-line load balancing. In *Online Algorithms: The State of the Art* (A. Fiat and G. Woeginger, eds). Springer LNCS 1441, 178–195, 1998.
7. Y. Azar and O. Regev. On-line bin-stretching. *Theor. Comput. Sci.*, 268:17–41, 2001.
8. Y. Bartal, H. Karloff and Y. Rabani. A better lower bound for on-line scheduling. *Infomation Processing Letters*, 50:113–116, 1994.
9. Y. Bartal, A. Fiat, H. Karloff and R. Vohra. New algorithms for an ancient scheduling problem. *Journal of Computer and System Sciences*, 51:359–366, 1995.
10. M.A. Bender and D.K. Slonim. The Power of team exploration: Two robots can learn unlabeled directed graphs. *35th Annual Symp. on Foundations of Comput. Sci.*, 75–85, 1994.
11. A. Blum and P. Chalasani. An online algorithm for improving performance in navigation. *SIAM Journal on Computing*, 29:1907–1938, 2000.
12. A. Blum, P. Raghavan and B. Schieber. Navigating in unfamiliar geometric terrain. *SIAM Journal on Computing*, 26:110–137, 1997.
13. T.C.E. Cheng, H. Kellerer and V. Kotov. Semi-on-line multiprocessor scheduling with given total processing time. *Theor. Comput. Sci.*, 337:134–146, 2005.
14. X. Deng, T. Kameda and C.H. Papadimitriou. How to learn an unknown environment I: the rectilinear case. *Journal of the ACM*, 45:215–245, 1998.
15. M. Englert, D. Özmen and M. Westermann. The power of reordering for online minimum makespan scheduling. *Proc. 49th Annual IEEE Symposium on Foundations of Computer Science*, 603–612, 2008.
16. U. Faigle, W. Kern and G. Turan. On the performance of on-line algorithms for partition problems. *Acta Cybernetica*, 9:107–119, 1989.
17. W. Feller. An Introduction to Prob. Theory and its Applications. John Wiley& Sons, 1968.

18. R. Fleischer and M. Wahl. Online scheduling revisited. *J. of Scheduling*, 3:343–353, 2000.
19. P. Fraigniaud, L. Gasieniec, D.R. Kowalski and A. Pelc. Collective tree exploration. *Networks*, 48:166–177, 2006.
20. G. Galambos and G. Woeginger. An on-line scheduling heuristic with better worst case ratio than Graham's list scheduling. *SIAM Journal on Computing*, 22:349–355, 1993.
21. R.L. Graham. Bounds for certain multi-processing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.
22. T. Gormley, N. Reingold, E. Torng and J. Westbrook. Generating adversaries for request-answer games. *Proc. 11th ACM-SIAM Symposium on Discrete Algorithms*, 564–565, 2000.
23. D.S. Hochbaum and D.B. Shmoys. Using dual approximation algorithms for scheduling problems: Theoretical and practical results. *Journal of the ACM*, 34:144–162, 1987.
24. S. Irani. Coloring inductive graphs on-line. *Algorithmica*, 11:53–72, 1994.
25. D.R. Karger, S.J. Phillips and E. Torng. A better algorithm for an ancient scheduling problem. *Journal of Algorithms*, 20:400–430, 1996.
26. R.M. Karp, U.V. Vazirani and V.V. Vazirani. An optimal algorithm for on-line bipartite matching. *Proc. 22nd Annual ACM Symposium on Theory of Computing*, 352–358, 1990.
27. H. Kellerer, V. Kotov, M.G. Speranza and Z. Tuza. Semi on-line algorithms for the partition problem. *Operations Research Letters*, 21:235–242, 1997.
28. A. López-Ortiz, S. Schuierer. On-line parallel heuristics, processor scheduling and robot searching under the competitive framework. *Theor. Comput. Sci.*, 310:527–537, 2004.
29. L. Lovász, M.E. Saks and W.A. Trotter. An on-line graph coloring algorithm with sublinear performance ratio. *Discrete Mathematics*, 75:319–325, 1989.
30. P. Raghavan and M. Snir. Memory versus randomization in on-line algorithms. *IBM Journal of Research and Development*, 38:683–708, 1994.
31. J.F. Rudin III. Improved bounds for the on-line scheduling problem. Ph.D. Thesis. The University of Texas at Dallas, May 2001.
32. J.F. Rudin III and R. Chandrasekaran. Improved bounds for the online scheduling problem. *SIAM Journal on Computing*, 32:717–735, 2003.
33. P. Sanders, N. Sivadasan and M. Skutella. Online scheduling with bounded migration. *Mathematics of Operations Reseach*, 34(2):481–498, 2009.
34. D.D. Sleator and R.E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202–208, 1985.